République du SENEGAL

Université Cheikh Anta DIOP

Ecole Supérieure Polytechnique

Centre de Dakar

Département Génie Informatique

Mémoire

Pour l'obtention du

DIPLOME D'INGENIEUR DE CONCEPTION

THEME :

Conception et implémentation dans MIMOSA des classes nécessaires pour la visualisation de l'évolution d'un modèle MIMOSA.

Présenté et soutenu par M. **Matar DRAME**

Encadreur :

M. **Mamadou NIANG** (ESP)

Lieu du stage : CIRAD

Maîtres de stage :

M. **Grégoire LECLERC** (CIRAD)

M. **Jean Pierre MULLER** (CIRAD)

M. **Alassane BAH** (ESP)

Juillet 2005

Table des matières

Introduction

La gestion de l'environnement et des territoires dans le monde a suscité beaucoup d'intérêts ces derniers temps dans le domaine de la recherche. Les politiques de développement consenties ont besoin d'une certaine valeur de vérité avant d'être mises en place car faudrait-il le rappeler il s'agit d'une mobilisation de beaucoup de ressources aussi bien financières qu'humaines. Ce qui donne l'intérêt d'une simulation avant toute prise de décision.

En outre, dans les systèmes environnementaux, il est toujours difficile voire impossible de mettre en place des formules mathématiques pouvant permettre de déduire des lois. Les équipes de recherche ont très vite apprécié l'apport de l'informatique à ce niveau, en ce qu'elle offre des concepts permettant de modéliser ce qu'on appelle systèmes complexes. L'informatique propose des systèmes, comme les agents entre autres, qui peuvent résoudre une bonne partie des problèmes environnementaux ; nous assistons depuis près de vingt ans à une explosion des systèmes multi agents après que l'intelligence artificielle ait montré beaucoup de limites. A partir de ce moment les chercheurs informaticiens ont commencé à réfléchir sur la mise en place d'outils de simulation qui seraient évidemment les bienvenues pour les environnementalistes, la gestion de l'environnement et des territoires nécessitant de disposer d'outils intégrés de représentation, d'exploration et d'explication des structures et processus spatiaux et temporels, physiques et sociaux.

La mise en place de ces outils va s'appuyer sur la modélisation mais aussi sur de puissants outils graphiques ce qui permettra aux utilisateurs non experts de pouvoir mener à bien des simulations et d'en tirer des conclusions. Ces simulations gagneraient par ailleurs à s'approcher le plus possible de la réalité. Ce qui donne une importance particulière à la visualisation.

C'est ainsi qu'une dizaine de laboratoires francophones se sont associés pour mettre en place une plate-forme générique de représentation des connaissances et de simulation : **MIMOSA**.

Dans la suite, nous présenterons MIMOSA et le travail qui nous est demandé. Ensuite, nous étudierons quelques plates-formes de modélisation et de simulation utilisées dans le domaine des systèmes multi agents, nous en présenterons les méthodes utilisées pour la visualisation au niveau du chapitre 2. Le chapitre 3 permettra de dégager les besoins du système et les fonctionnalités

mais également de présenter certaines vues du système de visualisation. L'implémentation vient en chapitre 4. En effet nous étudierons dans ce chapitre les outils nécessaires, tant du point de vue technique que mathématique. Nous conclurons en présentant des extensions possibles et des perspectives concernant la visualisation de l'évolution de modèle dans MIMOSA.

CHAPITRE 1 : PRESENTATIONS GENERALES

1. Présentation de la structure d'accueil (le CIRAD)

(Centre de coopération Internationale en Recherche Agronomique pour le Développement)

Le CIRAD est un institut français de recherche agronomique au service du développement des pays du Sud et de l'outre-mer français. Il privilégie la recherche en partenariat. Le CIRAD a choisi le développement durable comme ligne de force de son action à travers le monde. Cette démarche prend en compte les conséquences écologiques, économiques et sociales, à long terme, des processus de transformation des sociétés et des territoires du Sud. Le CIRAD intervient par des recherches et expérimentations, des actions de formation, d'information et d'innovation, et des expertises. Ses compétences relèvent des sciences du vivant, des sciences humaines et des sciences de l'ingénieur, appliquées à l'agriculture et l'alimentation, à la gestion des ressources naturelles et aux sociétés.

Vieille de plus de cent ans, la recherche agronomique tropicale française a connu un réel essor lorsque l'expansion et l'exploitation des productions tropicales ont été prises en considération dans l'économie nationale française, dans la seconde moitié du XXe siècle. La création du CIRAD fut l'aboutissement de ce processus. Comme tout organisme vivant, le CIRAD continue de s'adapter aux changements de la recherche et de l'aide au développement. Il participe ainsi pleinement, dans le cadre de sa mission, à l'équilibre de notre planète. La majorité des recherches menées par le CIRAD se fait en partenariat. Les programmes que les chercheurs conduisent avec leurs partenaires sont conçus et réalisés en commun. Les partenaires du CIRAD sont nombreux et variés : pouvoirs publics, instituts de recherche, universités, entreprises privées, groupements de producteurs, organisations professionnelles paysannes, entreprises du secteur agro-industriel, organisations non gouvernementales, etc. C'est d'ailleurs dans le cadre de ces partenariats que nous y effectuons notre stage de fin de cycle pour l'obtention du diplôme d'ingénieur de conception.

Ce stage s'effectue autour du projet MIMOSA, initié par le Professeur Jean Pierre MULLER, dans une de ses phases définie en collaboration avec le LGLIA (Laboratoire de Génie Logiciel et d'Informatique Appliquée) du département génie informatique de l'ESP (Ecole Supérieure Polytechnique). Le laboratoire comporte en son sein une équipe dénommée MSSC (Modélisation et Simulation des Systèmes Complexes). Nos travaux sont encadrés par

cette équipe qui constitue l'une des rares équipes de recherche travaillant en modélisation et simulation en Afrique.

2. Le projet MIMOSA (*Méthodes Informatiques Modélisation Simulations Agents*)

MIMOSA sera l'aboutissement d'un projet du même nom. MIMOSA devra être un logiciel générique de modélisation et de simulation écrit en Java. Le projet MIMOSA est l'un des plus gros projets de recherche-développement dans le milieu francophone. Il regroupe plus de dix (10) laboratoires implantés dans différents pays du monde. Pour une meilleure collaboration entre ces laboratoires, mais également du fait de la généricité il a été défini un modèle dit Modèle MIMOSA.

Pour la réalisation du projet il a été également défini deux grandes phases:

1. Spécification des API d'intégration capitalisant l'expérience des partenaires :
   - **a**. La couche 1 définira les besoins génériques minimaux pour exécuter des simulations de façon centralisée ou distribuée;
   - **b**. La couche 2 définira les API génériques nécessaires aux modélisateurs ;
   - **c**. La couche 3 définira les API génériques liés à la démarche de modélisation et à la conduite des simulations (analyse de sensibilité, exploitation des résultats, etc.).
2. Réalisation d'une plate-forme générique commune sur la base des spécifications avec son instanciation sur différents exemples afin de valider sa pertinence.

La réalisation de MIMOSA pose des défis scientifiques majeurs à la fois sur la représentation multi modèles, multi points de vue et multi échelles et sur la méthodologie de description et de conduite des simulations. Elle sera validée sur le papier à l'aide des exemples déjà développés par les partenaires.

Le but du projet MIMOSA est de réaliser une plate-forme de modélisation et de simulation générique offrant aux modélisateurs la possibilité de décrire des dynamiques variées, selon différents points de vue et d'en effectuer la simulation à des fins d'étude

d'impact. L'objectif général est de faciliter et d'améliorer la production et l'utilisation de modèles et simulations multi agents.

MIMOSA est construit sur :

- Une couche de base introduisant les composants (ou éléments), les composés (ou ensembles) et les relations pour la partie statique ; les états, les mesures et les événements pour la partie dynamique ;
- Un ensemble de « plug in » décrivent les composants, composés et relations particulières pour décrire un domaine déterminé.

MIMOSA constituera une plate-forme de modélisation et de simulation multi modèle. Le réel problème de la prise de décision auquel sont confrontés les dirigeants politiques peut voir un début de solution avec cet outil. Cependant il faut noter que le développement de MIMOSA nécessite beaucoup de ressources, ce qui explique son internationalisation et la démarche par « plug in » dont beaucoup sont déjà en marche, le reste étant en cours de développement. C'est dans ce contexte que le CIRAD s'est vu confier toute la partie qui définira les API génériques nécessaires aux modélisateurs**.** Et dans cette couche il nous a été proposé le sujet : ***Conception et implémentation dans MIMOSA des classes nécessaires pour la visualisation de l'évolution d'un modèle MIMOSA.***

3. Présentation du sujet (*Visualisation de l'évolution d'un modèle MIMOSA)*

Notre travail consistera à mettre en place un « plug in » MIMOSA qui permettra de visualiser l'évolution d'un modèle MIMOSA. Le graphisme constitue une large part dans ce travail mais auparavant il faudrait analyser tout ce que le modélisateur voudra visualiser et définir quelles sont les possibilités de visualisation. MIMOSA étant déjà entièrement développé en JAVA, il fallait continuer dans cette philosophie dite : « ***write once, run anywhere*** » qui signifie en français : Une seule écriture pour une exécution multi plates-formes.

Il conviendra pour une amélioration de ce qui existe déjà, de faire une étude sur l'état de la recherche dans les domaines de la modélisation, de la simulation et de la visualisation de modèles.

D'un point de vue analytique, notamment dans la spécification des besoins il faudra prendre en compte le fait que MIMOSA ne « connaît » que les notions de composants, composés et relations pour la description structurelle de modèles ; d'états, mesures et transitions pour la description dynamique de modèles. Par conséquent, l'analyse consiste dans une de ces facettes en la détermination de tout ce qui est concept structurel ou dynamique et de déterminer pour chaque concept structurel quels seront les paramètres variables dépendant de l'évolution du modèle (couleur, taille, …) mais également quels seront les paramètres dépendant de la vue (luminosité, contraste, …).

Visualiser un modèle MIMOSA consistera dès lors en la mise place de certaines relations entre composants, composé, relations du modèle et ceux de visualisation (exemple : un modélisateur peut décider qu'un agent sera représenté par une ellipse) mais surtout de définir les mesures et de les associer aux paramètres de visualisation variables (dépendant de l'évolution du modèle).

Nous procéderons de manière chronologique comme suit :

- Détermination des composants et des composés ;
- Spécification des paramètres variables (dépendant du modèle) ;
- Spécification des paramètres qui ne dépendent que de la vue ;
- Dans un second temps il faudra commencer à penser à l'implémentation car une grande partie de notre travail consistera à mettre en place des fonctions mathématiques permettant de gérer un monde graphique.

CHAPITRE 2 : LA PLATE-FORME MIMOSA

1. Contexte

1.1. Modélisation et Simulation

La modélisation et la simulation sont deux domaines en développement ces dernières années. En effet on assiste à une prolifération de laboratoires et d'équipes de recherche qui leur sont dédiés. La recherche y évolue très vite du fait non seulement de l'intérêt accordé par les experts mais aussi du fait de la croissance avérée des besoins. Ce sont des domaines qui présentent beaucoup d'avenir pour les jeunes chercheurs.

La transformation d'un besoin émergeant en la définition d'un système lui apportant une solution met en oeuvre de multiples activités intellectuelles faisant passer progressivement de concepts abstraits à la définition rigoureuse de produits. Il est nécessaire de s'appuyer sur des représentations tant du problème que de ses solutions possibles à différents niveaux d'abstraction pour appréhender, conceptualiser, concevoir, estimer, simuler, valider, justifier des choix et communiquer. C'est le rôle de la **modélisation**.

Un modèle est une simplification de la réalité qui permet de mieux comprendre le système modélisé. Il permet entre autre de visualiser le système comme il est ou comme il doit être.

La modélisation n'est pas un domaine exclusivement réservé aux informaticiens. Elle intéresse l'ensemble des chercheurs surtout ceux d'entre eux étudiant des systèmes complexes. En ce sens, la modélisation peut faire référence à toutes sortes de tâches interdisciplinaires bien différentes de la simulation numérique. La modélisation en Sciences de l'Ingénieur est une activité fédératrice par nature. En effet, elle fait appel aux diverses sciences de base de l'Ingénieur (Mathématiques Appliquées, Informatique, Thermique, Electricité, Résistance de matériaux, Mécanique des structures, …). La recherche en modélisation doit intégrer rapidement les nouveaux concepts et résultats issus de tous ces domaines, les adapter à ses propres besoins, puis les insérer dans des logiciels de plus en plus complexes. Ces logiciels de modélisation concentrent aujourd'hui le savoir et le savoir faire des organismes de recherche publique, ainsi que celui des entreprises, pour les rendre facilement utilisables par les décideurs (non experts dans le domaine de la modélisation pour la plupart).

La mise en place d'un modèle doit permettre de dégager des formalisations précises et détaillées qui peuvent être implémentées de manière logicielle dans un monde virtuel. C'est la **simulation**.

Les systèmes à simuler, étant de plus en plus complexes, l'utilisation des systèmes multi agents n'est pas seulement un besoin académique mais une réelle nécessité technique. Ces besoins couplés à une puissance de calcul énorme des nouveaux matériels impulsent la création de logiciels de modélisation et de simulation de plus en plus complets pour des systèmes jugés complexes.

Un des rôles de l'informaticien sera de concevoir de tels logiciels. C'est ainsi que des équipes de recherche se sont formées au niveau des universités et des organismes de recherche pour la réalisation de ces outils. Ces derniers sont nombreux, mais leur utilisation reste très limitée. Nous en citerons CORMAS, MADKIT, REPAST qui du reste sont les plus utilisés dans le domaine.

- **CORMAS (http://cormas.cirad.fr)**: CORMAS est un logiciel de modélisation et de simulation multi agent, spécialisé dans le domaine des ressources renouvelables. Il a été développé par les équipes de recherche du CIRAD. C'est un logiciel entièrement développé avec le langage SMALLTALK sous la plate-forme VISUALWORKS. CORMAS est très évolutif. En effet nous avons chaque année une nouvelle version améliorée du logiciel. Il est définit autour de trois modules :
    - Un premier module permet de définir les entités du système à modéliser ;
    - Le second module concerne le contrôle de la dynamique globale ;
    - Un troisième module permet de définir une observation de la simulation selon des points de vue ;

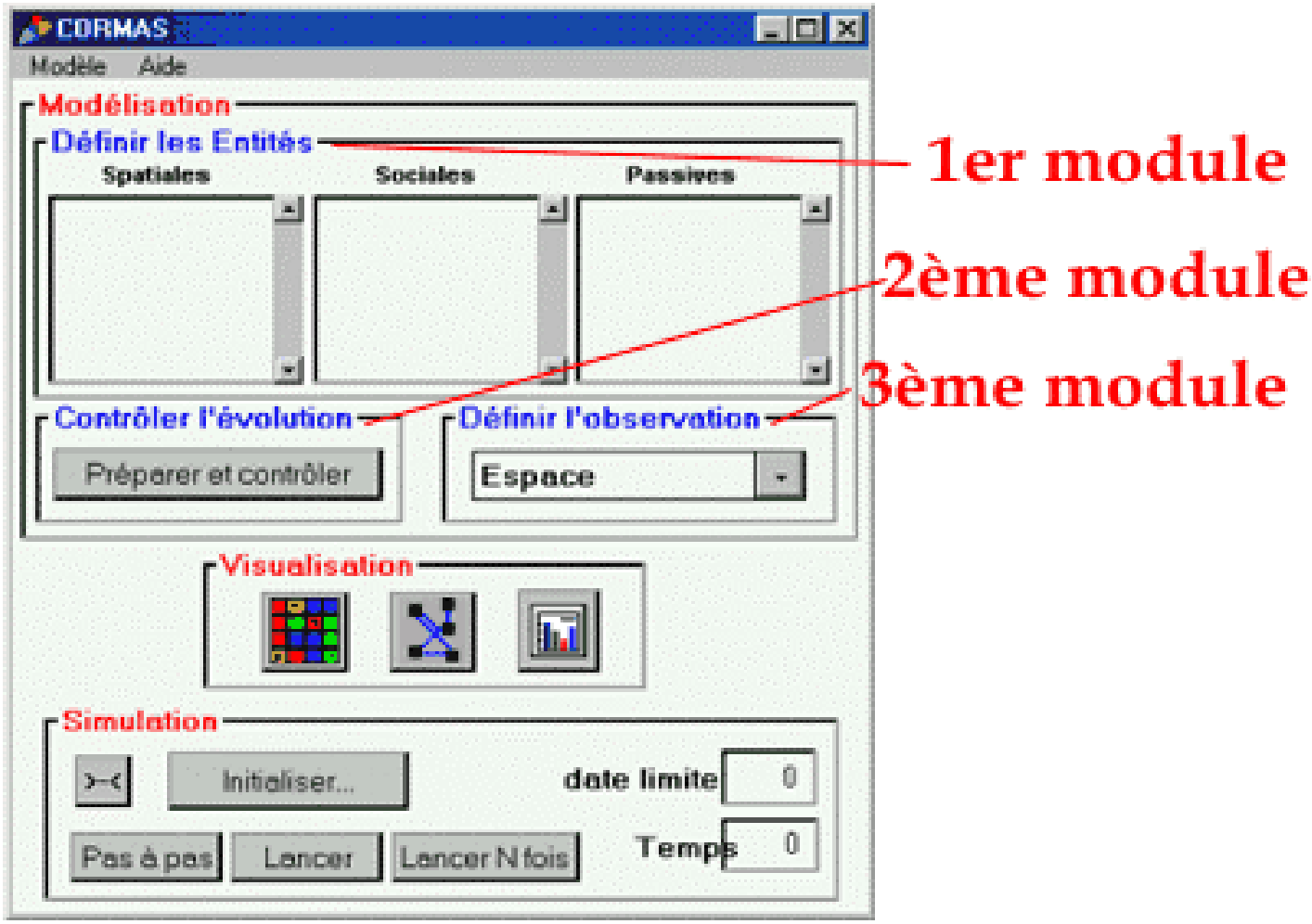


Figure 1: Cormas

- **MADKIT (www.madkit.org)** : Au même titre que CORMAS, MADKIT est un logiciel de modélisation et de simulation, il a été développé par une équipe de recherche sous la direction du Professeur Jacques FERBER (un spécialiste des systèmes multi agents). MADKIT est développé en JAVA, et nous avons actuellement des versions pouvant tourner dans la plupart des systèmes d'exploitation.

  Le modèle AGR (Agent Rôle Groupe), à partir duquel est construit MADKIT est fondé sur une interaction entre groupes, rôles et agents (notions propres aux systèmes multi agents) :

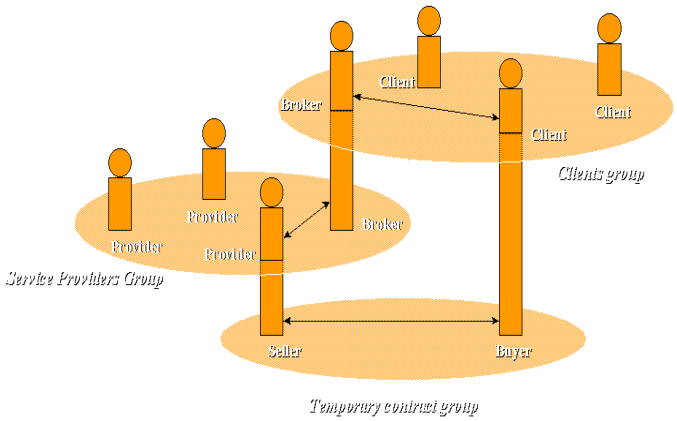


Figure 2: MADKIT

MADKIT peut fonctionner de manière distribuée. Il introduit aussi la notion de « communauté », c'est à dire d'application distribuée dans laquelle un ensemble d'agents peuvent se réunir et partager des applications et des documents. Il est ainsi possible de réaliser facilement de nombreuses applications distribuées: chat, partage de documents, jeux distribués, etc.

- **REPAST (http://repast.sourceforge.net/)** : C'est un projet américain de modélisation et de simulation multi agents. REPAST est développé à l'université de Chicago (aux USA). REPAST est un sérieux concurrent de MIMOSA, dans la mesure où il se voudrait, à l'instar de MIMOSA être le standard de modélisation et de simulation. C'est un projet Open Source, dirigé par des spécialistes américains. Comme MADKIT, REPAST est écrit en JAVA. L'un de ses points forts est l'intégration des systèmes d'informations géographiques.

1.2. Visualisation

a) Définition et avantages

La visualisation par ordinateur est l'utilisation d'images créées par ordinateur afin de comprendre des données de mesure ou de simulation. C'est un important nouveau domaine de recherche qui est partagé entre différentes sciences de l'ingénieur et l'informatique. Elle fournit des outils nécessaires pour extraire des informations en vue d'en tirer des connaissances.

La visualisation est un maillon très important de la connaissance scientifique. En effet, elle permet d'extraire l'information pertinente d'une masse de données et contribue à une meilleure compréhension du phénomène étudié.

Le développement d'outils de visualisation repose sur le fait que l'œil et le cerveau humains ont la capacité d'interpréter mentalement une image.

Aussi a-t-on l'habitude de dire qu' « **Un bon dessin vaut mieux qu'un long discours.** ».

Les bénéfices de la visualisation sont nombreux.

- La représentation visuelle a l'avantage d'être universelle, ne nécessitant pas l'apprentissage préalable d'une codification, à l'inverse des représentations codées utilisées par les communautés scientifiques. Elle est donc plus adaptée à l'échange d'informations entre les différentes communautés et à la vulgarisation scientifique.

- Dans un contexte de compétitivité accrue, la visualisation peut également jouer un rôle important dans la réduction des coûts de production, la simulation sur ordinateur remplaçant progressivement la réalisation de prototypes. En aéronautique la plupart des essais en soufflerie ont été remplacés par des simulations sur ordinateur. La visualisation permet de produire des animations permettant de voir l'écoulement de l'air le long du profil d'un appareil et de détecter les éventuelles anomalies de conception avant la réalisation d'un prototype. Ainsi l'apport de la visualisation va de l'exploration interactive de grands volumes de données à la conception et à la simulation de produits en passant par l'entraînement sur simulateur (simulateurs de vols, simulateurs chirurgicaux, ...).

b) <u>Principe</u> :

Le principe consiste en la représentation par des objets graphiques d'éléments non graphiques. La visualisation consiste à calculer une représentation visuelle de l'information de façon à en permettre une analyse rapide et efficace.

Son principe est basé sur le fait que notre système visuel est ce que nous avons de plus puissant en matière d'interprétation de l'information. Outre la capacité d'interpréter les images, nous avons également une faculté naturelle à reconstituer une information tridimensionnelle à partir d'une séquence animée d'images 2D.

La clé du problème est alors de définir les éléments visuels les plus adéquats pour représenter l'information et de produire les algorithmes qui permettront de calculer automatiquement une telle représentation à partir des données brutes. Il s'agit d'un problème difficile car les données originales sont souvent volumineuses, multidimensionnelles, multimodales et de structure complexe.

Enfin les algorithmes doivent prendre en compte un certain nombre de spécificités liées à leur utilisation : volume de données à traiter très important (de un méga-octets à plusieurs tera-octets), besoin d'interactivité, données réparties, interface avec les Systèmes de Gestion de Bases de Données, adaptation à l'Internet, fonctionnement en mode client serveur, support Web.

Les avantages de la visualisation et le caractère scientifique de son principe ont fait que les plates-formes de modélisation et de simulation intègrent de puissants systèmes de visualisation. Nous étudierons ces systèmes pour CORMAS et MADKIT.

c) La visualisation dans quelques plates-formes existantes :

L'un des points, forts ou faibles selon le cas, des plates-formes de modélisation est leur système de visualisation. Nous étudions dans cette partie la façon dont certaines plates-formes (notamment CORMAS et MADKIT qui se trouvent être les plus connus dans le milieu francophone) ont traité la visualisation des modèles.

i. CORMAS

L'un des modules qui constituent CORMAS est le système de visualisation. CORMAS définit dans son modèle les notions d'entités spatiales, d'entités sociales et d'entités passives.

Concernant, la visualisation, CORMAS a mis à disposition une grille qui représente l'espace dans lequel évoluera le modèle (c'est-à-dire les entités sociales et les entités passives). La grille est constituée de cellules adjacentes qui sont elles même des entités spatiales. CORMAS introduit la notion de « probe » (observateur) dans ces modèles, ce

qui permet aux modélisateurs de suivre l'évolution de certains paramètres du modèle lors de la simulation. Toutefois il faut écrire du code (programmer en SMALTALK) pour faire évoluer les paramètres variables de la vue dépendant de l'évolution du modèle.

Les seuls éléments de visualisation disponibles sont des polygones dont on spécifie à sa guise le nombre de côtés. CORMAS ne travaille qu'en dimension 2. Dans CORMAS, c'est la bibliothèque qui limite les fonctionnalités, cette dernière n'est pas très riche. De ce fait les possibilités de visualisation sont réduites. Ce qui est de moins en moins vrai avec le couplage « statique » avec des système d'informations géographiques (MAPINFO, ARCVIEW)

ii. MADKIT

MADKIT met à disposition un puissant outil d'édition et de visualisation de modèles : SEDIT.

SEDIT est un outil écrit en java, capable de tourner en « stand alone ». En effet, une version de SEDIT a été publiée sur Internet au mois de mars 2005. SEDIT met à disposition les rudiments nécessaires à la visualisation de modèles dans MADKIT. Dans SEDIT, un formalisme est défini dans un fichier XML. Ce fichier doit être associé à une classe JAVA pour spécifier la visualisation et l'animation du modèle.

Il faut noter le lien fort qui existent entre le modèle et sa représentation graphique, mais aussi le besoin d'une connaissance de JAVA pour visualiser.

Un lourd travail a été mené par les chercheurs pour la mise en place d'outils de visualisation de modèles. Cependant, l'objectif de MIMOSA est non seulement d'offrir une visualisation proche de la réalité mais aussi MIMOSA veut s'appuyer véritablement sur le principe MVC (Modèle Vue Contrôleur). MIMOSA se propose de faire un traitement particulier pour la visualisation de systèmes d'informations géographiques, qui seront considérés comme des modèles MIMOSA.

d) La visualisation dans MIMOSA :

La visualisation de modèles est très importante dans la mesure où elle permet à un modélisateur de suivre l'évolution de son modèle. Elle consiste en la représentation par des

objets graphiques (visuels) des éléments d'un modèle donné. Une plate-forme de modélisation « toute faite » sans système de visualisation n'en ait pas une.

La visualisation de modèle MIMOSA devra s'appuyer sur le principe MVC (Modèle Vue Contrôleur) c'est-à-dire que la vue est totalement différente du modèle visualisé.

Visualiser un modèle MIMOSA, c'est donner une représentation graphique de ce dernier. Pour ce faire, il faudra disposer d'un ensemble d'objets qui permettront de représenter un modèle MIMOSA et d'en montrer l'évolution lors des simulations. La vue sera elle-même un modèle MIMOSA, c'est ainsi qu'il faut intégrer les notions de base définies dans MIMOSA dans la conception de ces objets graphiques.

L'articulation entre modèles MIMOSA nous permettra de réaliser des correspondances entre les modèles MIMOSA et les vues : l'objet de l'API que nous devons mettre sur pied.

2. Les concepts MIMOSA

MIMOSA est un logiciel de modélisation et de simulation multi modèle.

Du fait d'un besoin de généricité, il a été nécessaire de partir d'une définition « générique » du concept modèle. C'est ainsi que dans MIMOSA, on part du principe que :

- Tout modèle est un discours sur l'expérience ;
- Ce discours s'appuie sur des concepts (Un modèle est un ensemble de concepts, dans MIMOSA);
- Ce discours se donne les moyens pour se dire ; ces moyens sont des formalismes, si toutefois ils ont une forme (c'est-à-dire respectent une syntaxe) ; Ce discours respecte donc un certain formalisme, une syntaxe ;

Faire un outil générique de modélisation reviendrait à définir les discours (c'est-à-dire les modèles) et les moyens du discours (c'est-à-dire les formalismes).

Un autre principe est que tout modèle peut se décrire autour de notions de base élémentaires :

- Les notions structurelles : composant (component), composé (compound), relation.
- Les notions dynamiques : état, évènement, mesure, fonction de transition.

Dans MIMOSA, nous avons quatre niveaux de discours représentés comme suit :

a. le niveau asémantique qui sert de base à la plate-forme MIMOSA pour décrire les différents formalismes. Ce niveau met à disposition les notions de base : composant, composé, relation.

b. le niveau des formalismes directement programmables dans la plate-forme MIMOSA pour les différents besoins spécifiques. Un certain nombre d'éléments pour parler de l'espace, des automates cellulaires, du temps,… ont été déjà définis. Nous ajouterons les éléments pour parler de vues (d'objets graphiques,…). Ce niveau met à disposition ce qu'on appelle les méta concepts puisqu'il s'agit du formalisme pour représenter les modèles et non les modèles.

c. le niveau des types : Il permet de décrire les catégories d'objets dont on va parler : les types de composants, les types de composés, les types de relations. Ce qui nous amène déjà au modèle proprement dit. Ce niveau met donc à disposition les moyens pour décrire un modèle.

d. le dernier niveau permet au modélisateur d'utiliser les types qu'il s'est donné précédemment pour construire son modèle. Il pourra par la suite exécuter la simulation du modèle défini.

Les deux premiers niveaux sont accessibles par programmation, le premier niveau étant figé une fois pour toute, le deuxième proposant un certain nombre de formalismes prédéfinis.

Les deux derniers niveaux sont accessibles au(x) modélisateur(s) via une interface graphique dédiée.

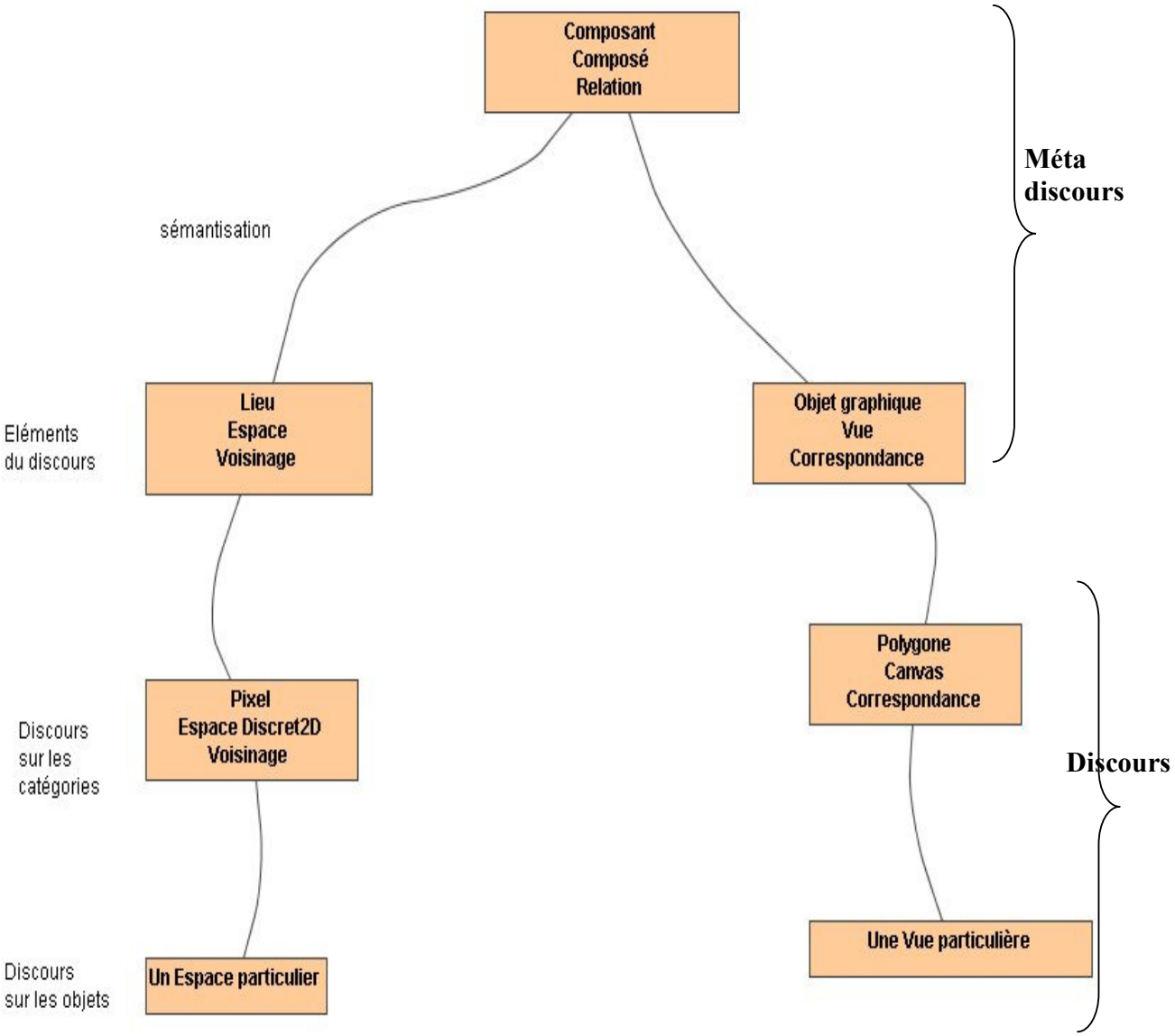


Figure 3:Exemple de modèles MIMOSA

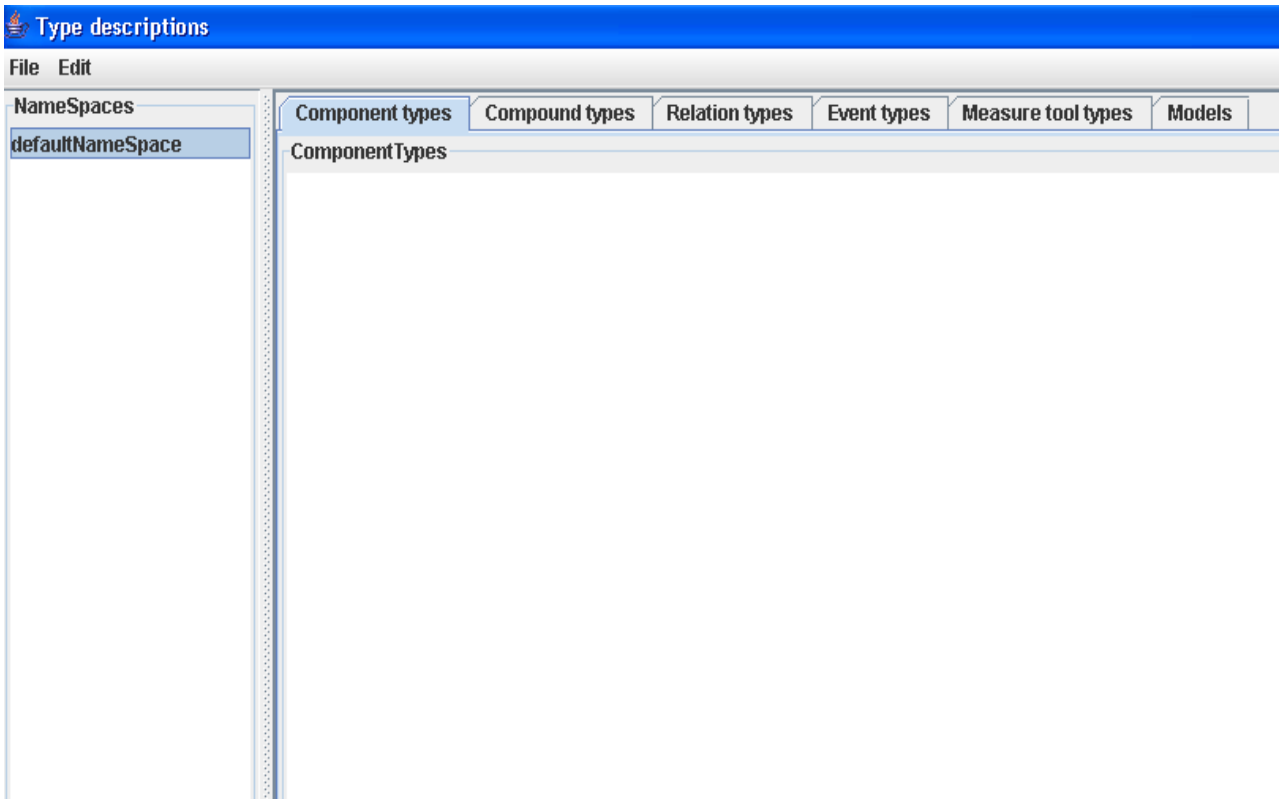


Figure 4: Interface graphique de MIMOSA

Pour décrire un modèle, on a besoin de formalismes. En effet ces formalismes sont prédéfinis dans MIMOSA au niveau 2 (b). A ce niveau, les formalismes disponibles sont décrits à travers des notions de bases. Ces notions doivent permettre de décrire aussi bien la structure du modèle que la dynamique de celui-ci. C'est pourquoi MIMOSA met à disposition deux groupes de notions :

- *Les notions structurelles pour décrire la structure ; ce dont on parle*
- *Les notions dynamiques pour décrire la dynamiques ; les processus*

2.1. les notions structurelles

Nous avons comme notions structurelles les concepts de composants, composés et relations :

- les composants : sont des entités non décomposables.
- les composés : sont des agrégats de composants

- les relations : nous avons
    * les relations « intra composé »
    * les relations « inter composés »
    * les relations entre un composé et un composant

Ces notions n'ont pas de sémantique en elles même, mais on considère que tout type de formalisme peut se décrire à partir de ces notions de base.

Exemple :

Un espace peut être représenté comme un ensemble de lieux muni d'une topologie, par exemple sous la forme d'une relation de voisinage. Les composants sont alors les lieux, le composé est l'espace dont les composants sont nommés par des coordonnées ou des noms de lieu selon l'usage. La relation interne est la relation de voisinage. Les relations entre composés peuvent permettre de décrire les relations possibles entre deux espaces, notamment les changements de coordonnées.

Il est donc possible de décrire les notions avec lesquelles construire un discours sur l'espace. A partir de là il devient possible de construire le discours proprement dit : **si on a des formalismes pour décrire des espaces, on peut décrire des espaces.**

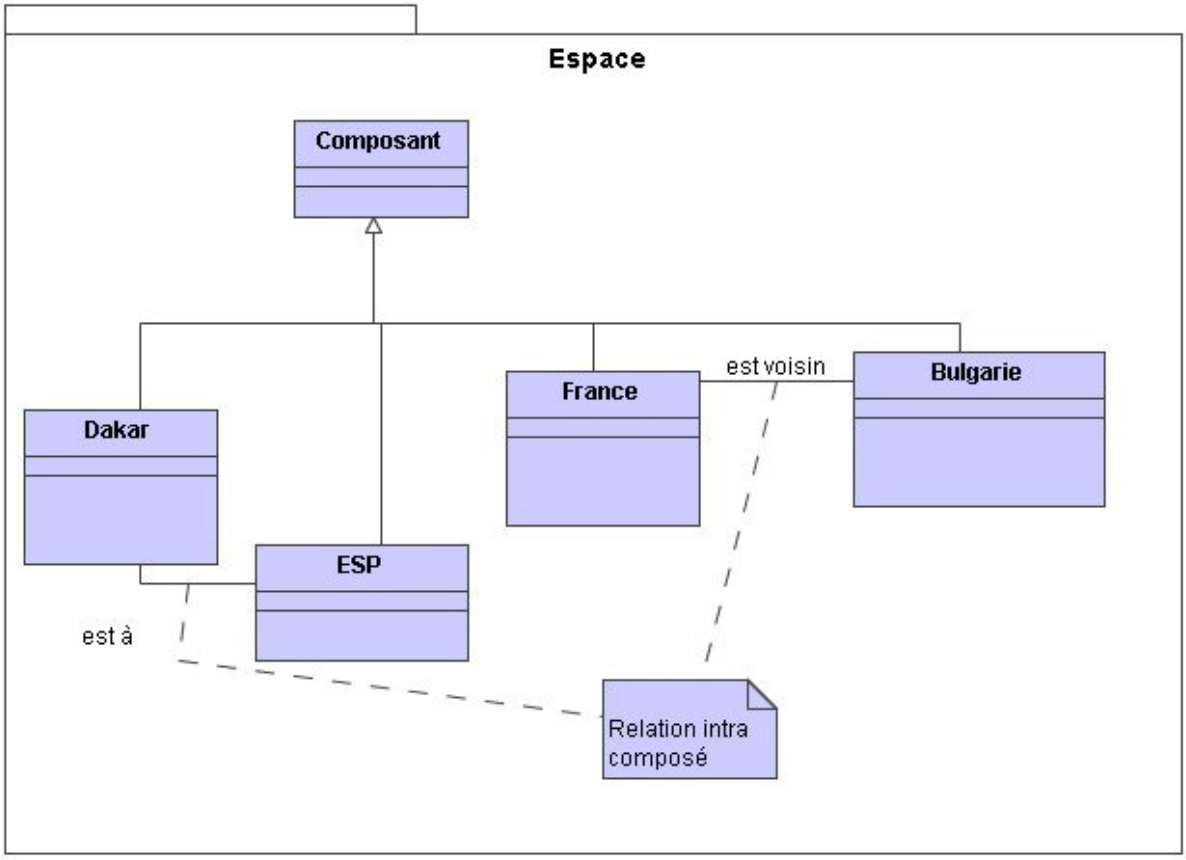


Figure 5: Diagramme d'objet du Modèle Espace selon MIMOSA

Articulation entre modèles MIMOSA

La notion d'articulation permet à MIMOSA d'être multi modèle, au sens où cette notion lui permet de relier deux modèles différents. Contrairement à l'intégration qui supposerait un discours universel avec lequel on pourra exprimer tout ce qu'on voudrait dire.

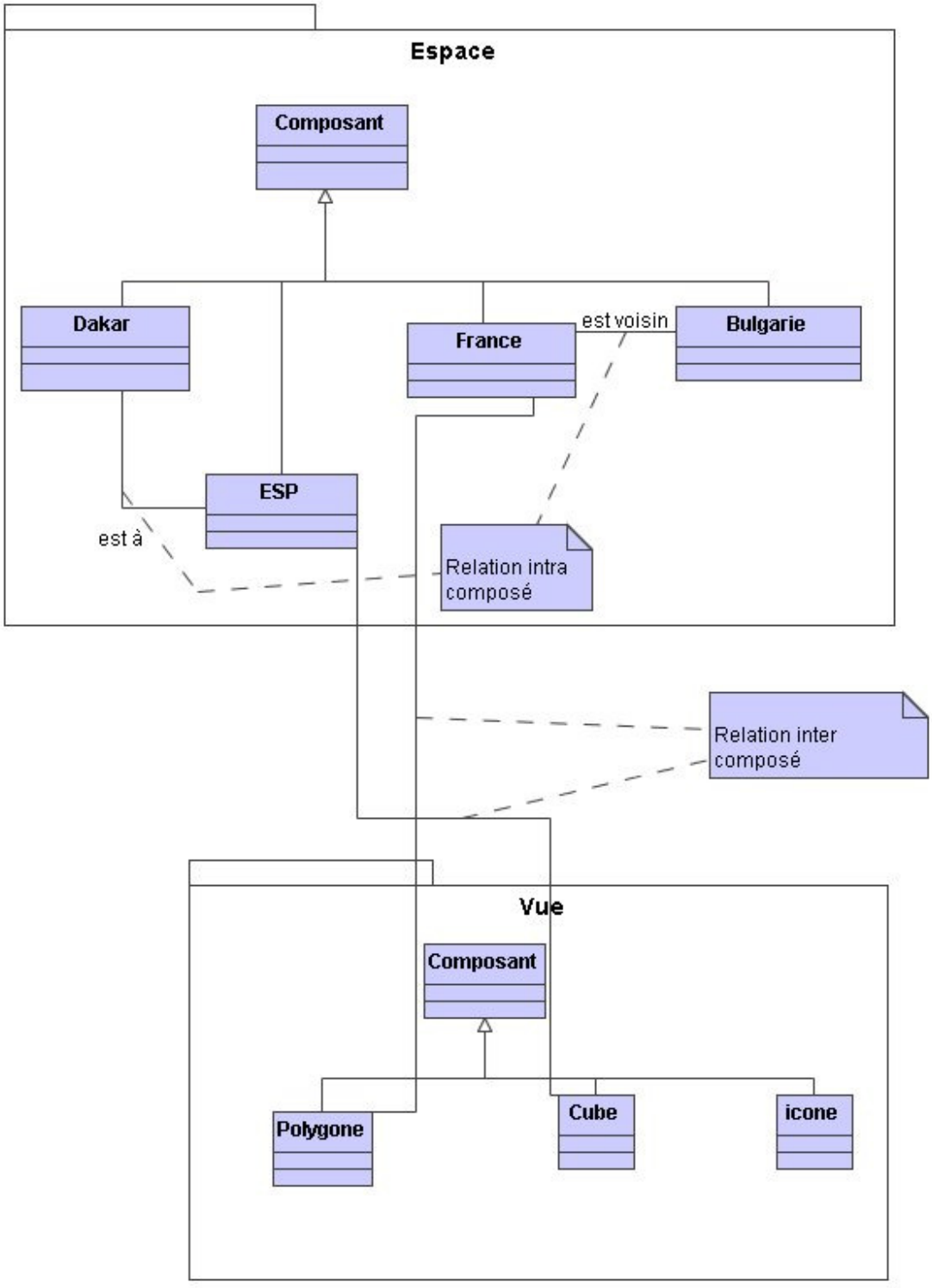


Figure 6: Articulation entre deux modèles MIMOSA

Mise en œuvre de ces concepts :

La mise en œuvre de ces concepts est faite autour de classes abstraites java. La description de ces classes est la suivante :

Component : La classe qui correspond aux parties des formalismes qui dénotent les concepts individuels unitaires, appelé aussi des instances de composants ou des composants.

Compound : La classe qui correspond aux parties des formalismes qui dénotent les concepts individuels unitaires, appelé aussi des instances de composés ou des composés.

ComponentType : La classe qui correspond aux parties des formalismes qui dénotent les concepts catégoriels composant, appelé aussi les types de composants ;

CompoundType : La classe qui correspond aux parties des formalismes qui dénotent les concepts catégoriels composé, appelé aussi les types de composés ;

Name : Une interface dont, les implémentations doivent définir ce qu'est un nom pour le composé.

A l'exception de Component, chaque classe dispose d'un nom (**getName ()** qui rend un objet de type Name). Ce n'est pas le cas des composants car on peut considérer que le nom n'est rien d'autre que celui qui permet de le distinguer dans le composé. De ce fait, on peut dire qu'il en possède. Les concepts individuels se veulent systématiquement des instances des concepts catégoriels, en ce sens ils offrent le type (**getType ()** qui renvoie le type).

Les composants définissent les moyens d'accéder aux composés (**getCompound()**), aux autres composants de son propre composés (**getComponent(Name)**) ainsi qu'aux composants et composés avec lesquels le composant est en relation (**getRelatedComponents(String), getRelatedCompounds(String)**), De la même manière chaque composé permet d'accéder (**getComponent(Name)**), d'ajouter (**addComponent(Name, Component)**), de modifier (**setComponent(Name,Component)**) et de détruire (**removeComponent(Name)**) ces composants.

Il est également défini un ensemble de types de relation :

IntraCompoundRelationType : la classe qui correspond aux parties des formalismes qui dénotent les types de relations entre composant d'un même composé;

InterCompoundRelationType : la classe qui correspond aux parties des formalismes qui dénotent les types de relations entre composant de composé distinct;

ComponentCompoundRelationType : classe qui correspond aux parties des formalismes qui dénotent les types de relations entre composant et composé (en plus de l'appartenance) ;

Anisi que leurs instances respectives : **IntraCompoundRelation, InterCompoundRelation, ComponentCompoundRelation.** Ces classes sont également munies d'un nom (**getName()**) et surtout d'une fonction qui permet de transformer un élément dans un ou plusieurs autres. Cette fonction dépend du formalisme qui lui donne un sens.

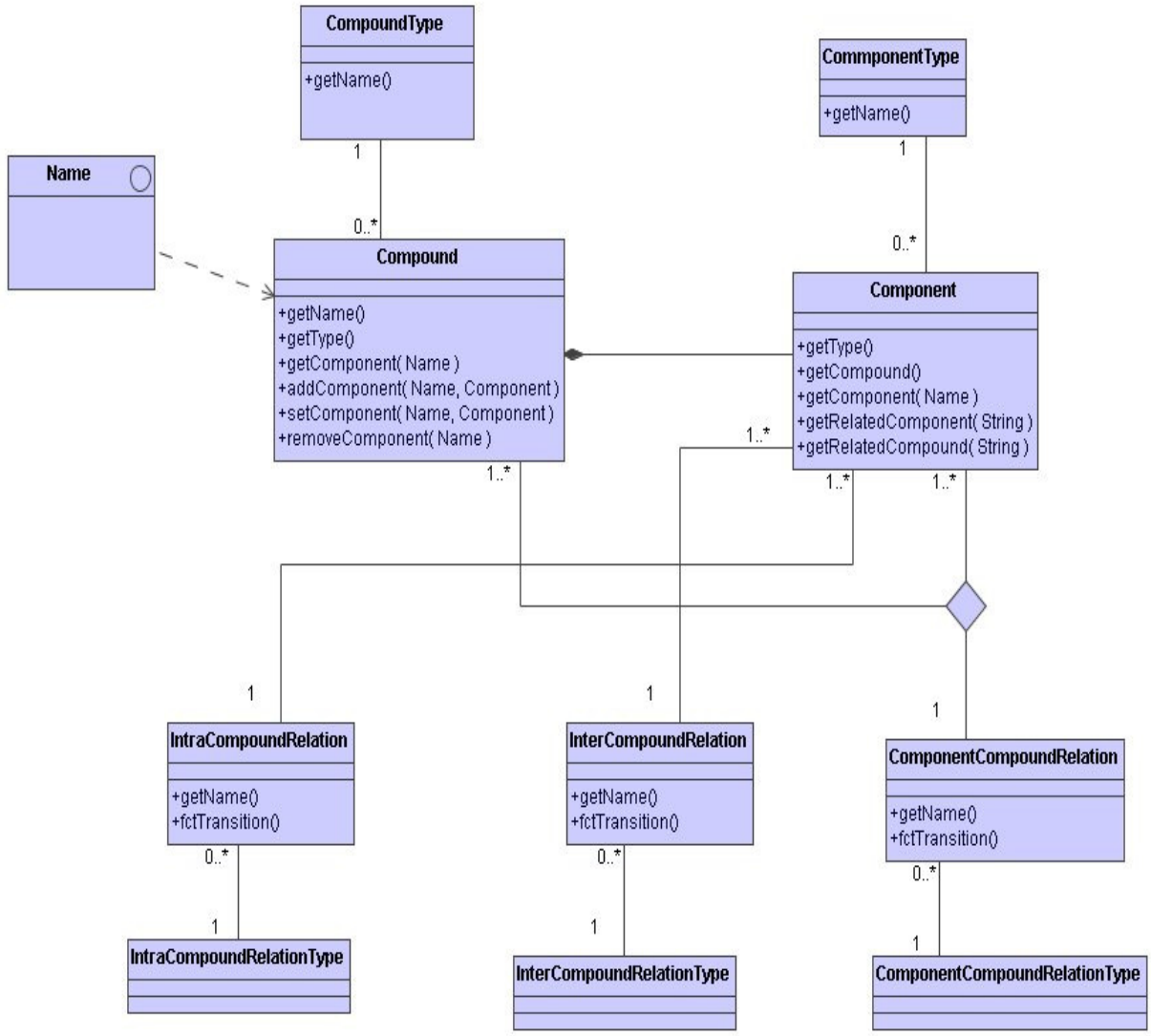


Figure 7: Diagramme de classe (implémentation) de base de MIMOSA

2.2. les notions dynamiques

Concernant la dynamique des systèmes, MIMOSA a défini les notions suivantes :

- **Les états** : Chaque composant ou composé peut être associé à un état qui permet de décrire l'état dynamique ;

- **Les évènements :** Ils imposent les changements d'état. A ce titre ils sont traités et générés par les composants et composés ;
- **Les mesures** : Elles donnent des informations sur l'état des composants ou composés. Pour les composés, la mesure minimale est l'accès aux composants ;
- **Les fonctions de transition :** Elle permettent de décrire comment un composant ou un composé réagit à un évènement.

Les états d'un composé sont en général, les états de l'ensemble de ses composants à un moment donné, les évènements étant l'ajout et le retrait d'un composant. Une mesure possible est l'accès au composant mais on peut en imaginer d'autre selon le système à modéliser.

Notons l'association entre les concepts catégoriels, l'ensemble des évènements et mesures qu'ils définissent, ainsi que l'état et la fonction de transition.

Mise en œuvre

La mise en œuvres des concepts dynamiques est faite autour de méthodes mises à disposition au niveau des classes décrivant les notions structurelles.

Les classes de la partie structurelles permettent de définir :

- L'ensemble des évènements que le concept accepte ;
- L'ensemble des mesures que l'on peut appliquer sur le concept ;
- La fonction de transition qui reçoit un évènement et le traite en fonction de l'état interne.
- La fonction de traitement de la mesure qui reçoit la spécification d'une mesure et rend le résultat de la mesure.

CHAPITRE 3 : ANALYSE ET CONCEPTION DU SYSTEME DE VISUALISATION DE MIMOSA

L'analyse et la conception sont faites autour du concept d'objet. En effet ce concept porte une philosophie de mise en place de logiciel adopté par la plupart des informaticiens. Le développement orienté objet comporte beaucoup d'avantages.

En outre, nous utilisons UML (Unified Modeling Language) pour décrire les différents aspects du système. UML met à disposition neuf diagrammes qui permettent de décrire tout système. Cependant, il n'est pas toujours nécessaire de représenter tous ces diagrammes pour mettre en place un logiciel. Nous utilisons dans le cadre de notre étude les diagrammes de cas d'utilisation et de classes.

1. Analyse des besoins du système

MIMOSA est un projet ambitieux, ses objectifs sont énormes et demandent un lourd travail d'analyse et de conception.

Concernant la visualisation, le système doit respecter les points suivants :

- Il doit être le plus proche de la réalité ;
- Il doit présenter une large gamme de choix concernant les objets graphiques disponibles pour visualiser les modèles ;
- Il doit pouvoir tourner sur une grande parties des systèmes d'exploitation (MAC OS, WINDOWS, UNIX,…) ;
- La vue doit être indépendante du modèle. Ce dernier pouvant par ailleurs disposer de plusieurs vues ;
- Le système doit prendre en compte les systèmes d'information géographiques.

1.1. Les fonctionnalités du futur système

MIMOSA étant multi modèles, ses besoins en terme de visualisation sont nombreux. En effet le système de visualisation doit présenter les fonctionnalités suivantes :

- Définition claire et nette des paramètres dépendant du modèle (idem pour les paramètres ne dépendant pas du modèle) ;
- Le système doit permettre à un modélisateur de choisir un modèle dans la mémoire, le visualiser, en effectuer une simulation.

- Le système doit permettre de sauvegarder une vue et d'en effectuer le rechargement ultérieurement.
- Le système doit permettre au modélisateur de choisir dans une palette les objets graphiques qu'il veut utiliser pour visualiser son modèle.
- Le système doit permettre au modélisateur de dessiner un objet graphique et de l'ajouter à la palette de choix.

1.2. Les cas d'utilisation : Réalisation des objectifs des utilisateurs

Un cas d'utilisation est constituant de l'expression des besoins.

Il décrit comment un **acteur** externe au système exerce sur ce dernier une action dans un but précis, et la réponse dudit système. C'est donc un ensemble de séquences qu'effectue le système pour fournir un résultat observable pour un acteur.

Un cas d'utilisation peut avoir des relations avec d'autres. Les types de relations les plus utilisés sont :

« **Include** » : indique que le cas d'utilisation source contient aussi le comportement décrit dans le cas d'utilisation destination.

« **Extends** » : indique que le cas d'utilisation source étend (précise) les objectifs du cas d'utilisation destination.

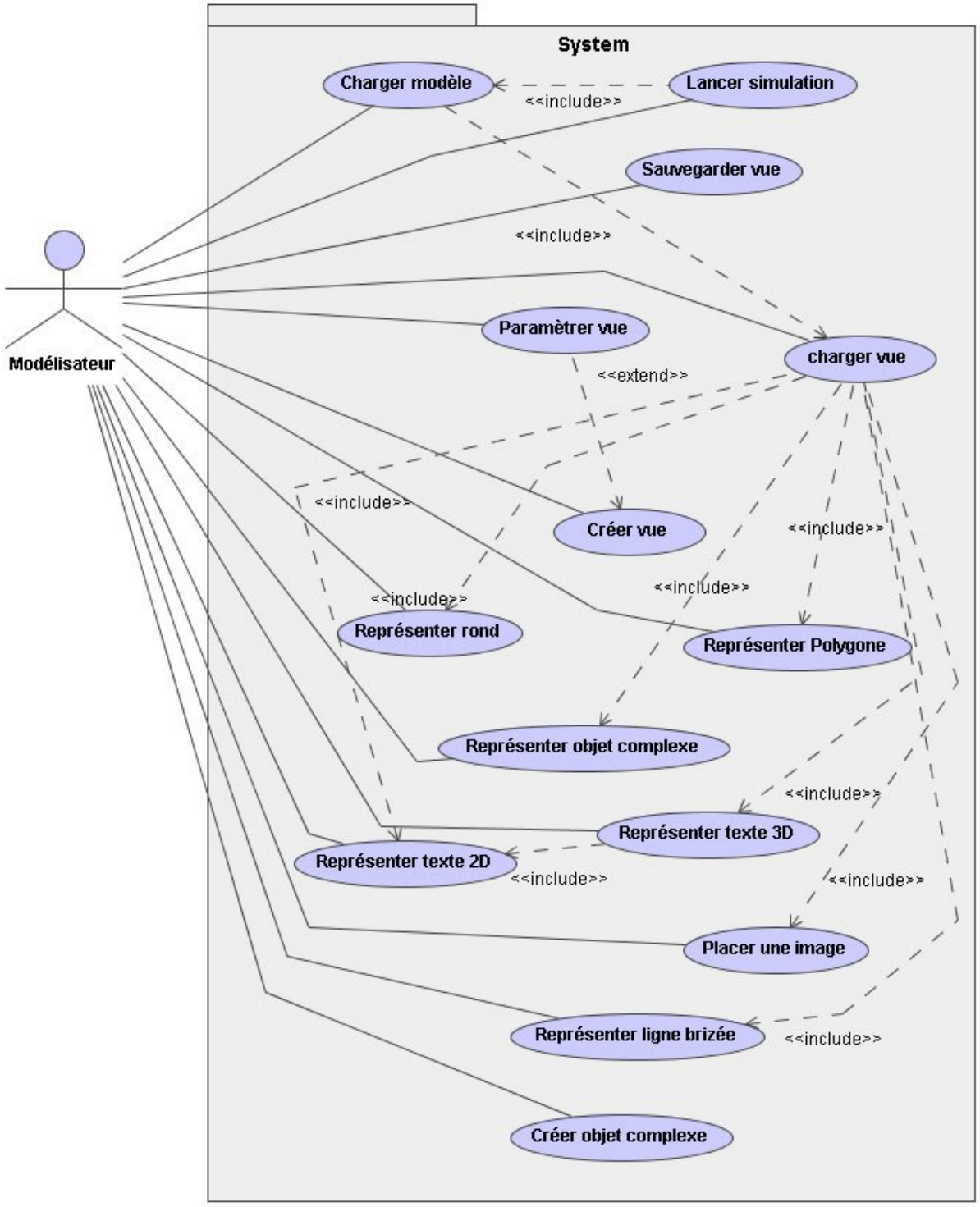


Figure 8: Diagramme de cas d'utilisation du système

L'étude de ces cas d'utilisation induit les vues dynamiques du système que nous présentons dans ce qui suit.

2. Vues dynamiques du système

Le système est défini autour des cas d'utilisation suivant : (nous en donnons une étude élaborée pour les plus important).

Dans notre étude, nous considérons que nous sommes en 3D ce qui implique l'utilisation des notion de caméra, de luminosité et de texture.

❖ **Paramétrer vue**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton paramétrer vue.

Pré condition : Un modèle est déjà chargé.

Post condition : La vue du modèle est réinitialisée.

Fin : Le cas d'utilisation se termine lorsque le système adopte la vue définit par l'utilisateur.

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur choisit de paramétrer la vue.	
	2. Le système demande d'indiquer la position de la camera
3. Le modélisateur indique la position de la camera.	
	4. Le système demande l'éclairage
5. Le modélisateur donne l'éclairage.	
	6. Le système demande la texture.
7. Le modélisateur donne la texture.	
	8. Le système charge la vue

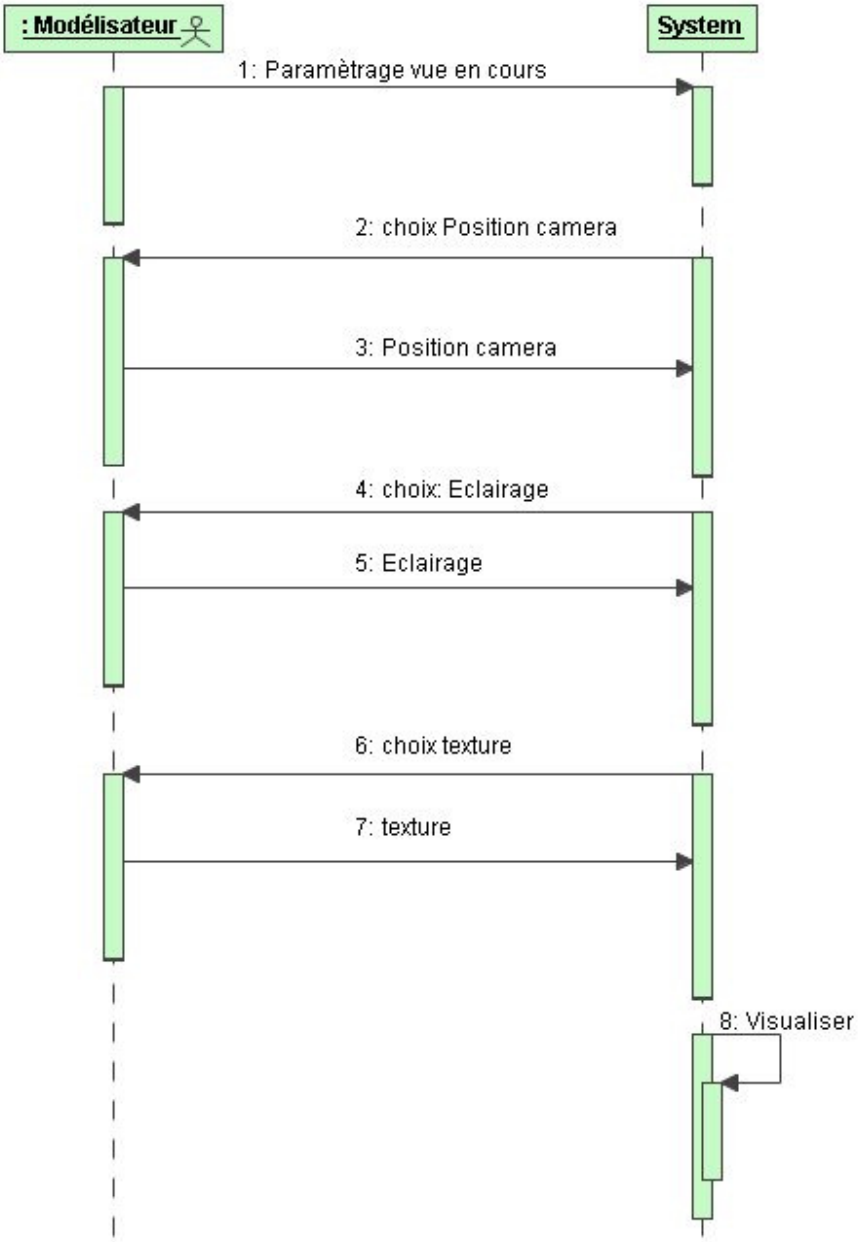


Figure 9 : Paramétrer vue, Diagramme de séquence du scénario nominal

❖**Charger un modèle**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton charger un modèle.

Fin : Le cas d'utilisation se termine lorsque le système affiche une vue du modèle.

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur choisit de charger un modèle.	
	2. Le système lui demande d'entrer le chemin d'accès du fichier correspondant au modèle.
3. Le modélisateur indique le chemin d'accès du fichier	
	4. Le système vérifie si le fichier est réellement un fichier représentant un modèle MIMOSA.
	5. Le système présente au modélisateur l'ensemble des composants, composés et relation du modèle choisi par ce dernier.
6. Le modélisateur associe à chaque élément du modèle (composant, composé, relation, …) un élément graphique.	
	7. Le système affiche pour chaque élément du modèle sa représentation graphique et deux listes l'une pour les paramètres variables des objets graphique, l'autre pour les mesures.
8. Le modélisateur fait les correspondances entre paramètre variables des objets graphiques et mesures des éléments du modèle.	

	9. Le système donne une vue du modèle.

Enchaînement alternatif :

Enchaînement au point 3 :

Le fichier spécifié n'est pas conforme MIMOSA.

Echec de chargement.

Reprise au point 3.

Représentation graphique

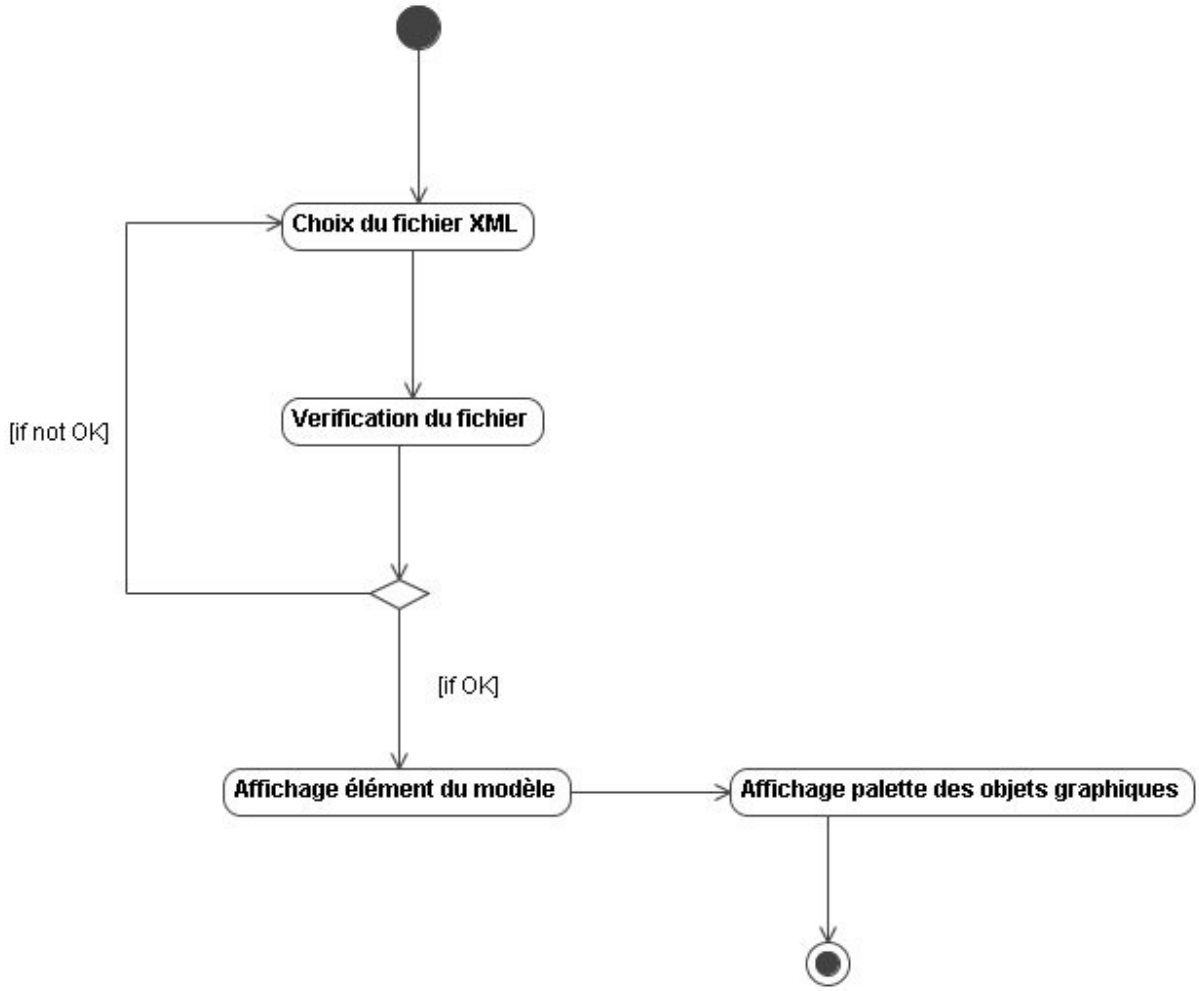


Figure 10: Diagramme d'activité qui montre les cas alternatifs et les erreurs

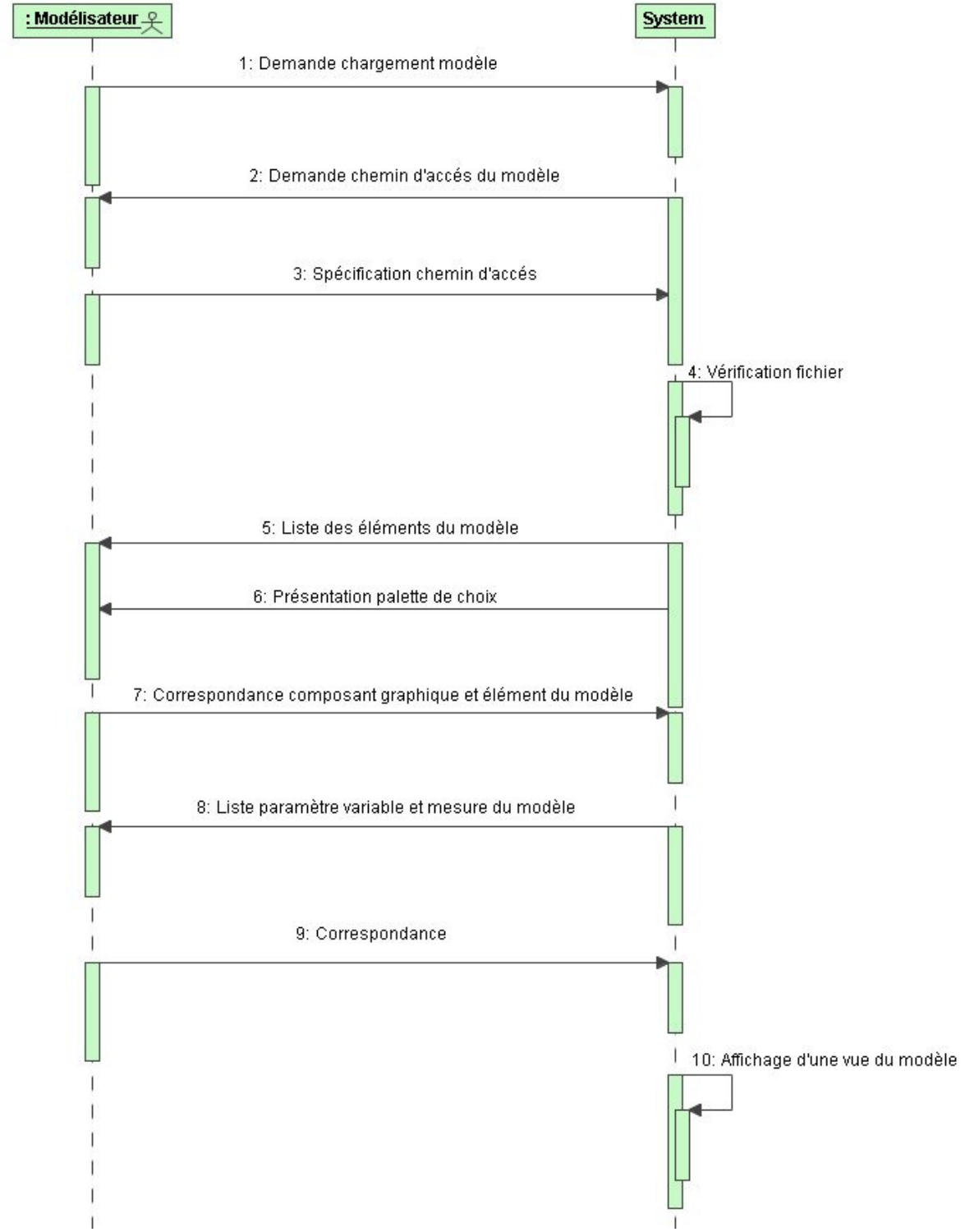


Figure 11: Charger modèle, Diagramme de séquence du scénario nominale

❖**Représenter un rond :**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton dessiner rond.

Pré condition : Une vue est déjà chargée.

Post condition : Un rond est positionné sur la vue.

Fin : Le cas d'utilisation se termine lorsque le système dessine un rond sur la vue

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur demande de dessiner un rond.	
	3. Le système demande d'enter les coordonnées du centre.
2. Le modélisateur clique sur la position du centre.	
4. Le modélisateur choisit la taille du rond avec la souris.	5. Le système affiche un rond.

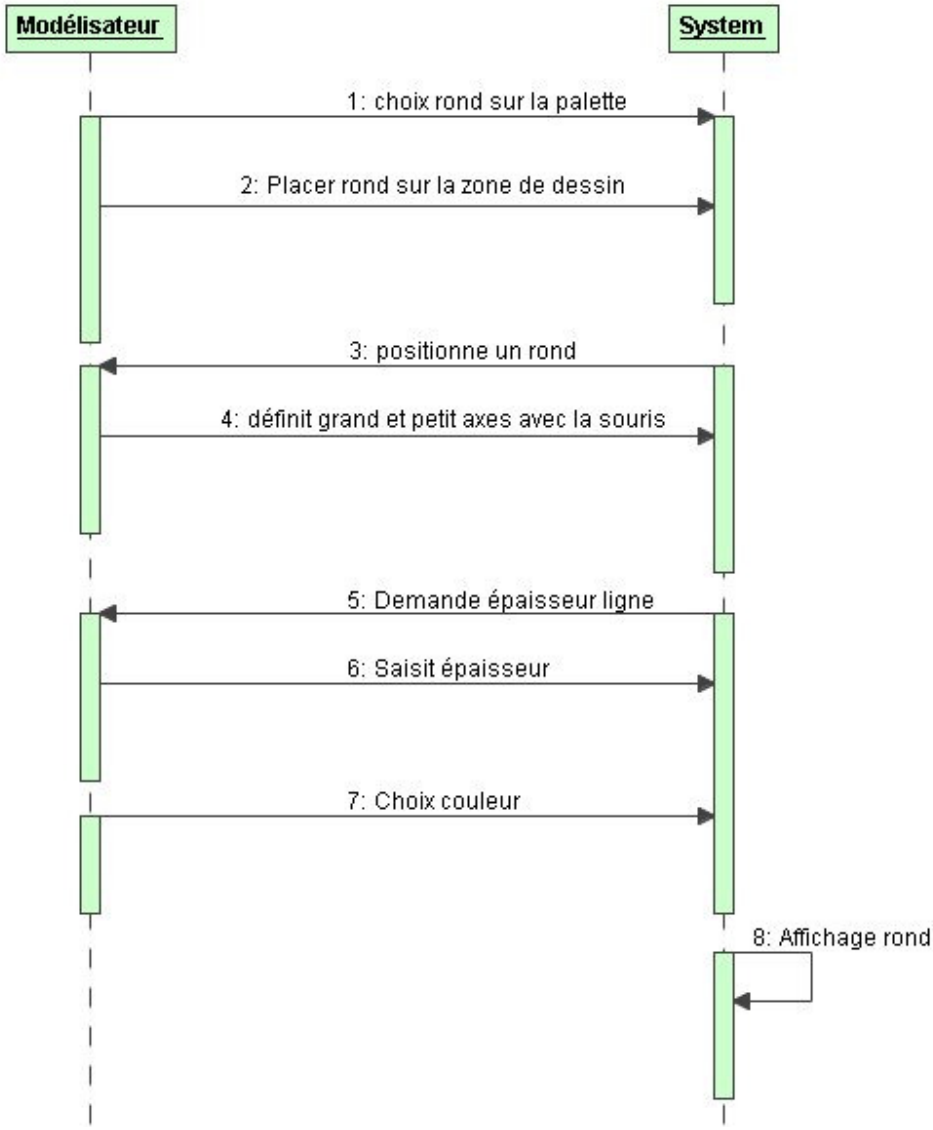


Figure 12: Cas d'utilisation : Représenter un rond, Diagramme de séquence du scénario nominal.

❖ **Représenter polygone :**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton dessiner polygone.

Pré condition : Une vue est déjà chargée.

Post condition : Un polygone est positionné sur la vue.

Fin : Le cas d'utilisation se termine lorsque le système dessine un polygone sur la vue.

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur demande de représenter un polygone.	**2**. Le système demande d'entrer les points composant le polygone.
3. Le modélisateur donne la liste des points en cliquant sur leurs positions.	4. Le système demande d'entrer les couleurs.
5. Le modélisateur spécifie les couleurs.	6. Le système demande d'entrer l'épaisseur
7. Le modélisateur spécifie l'épaisseur.	8. Le système dessine un polygone dans la vue.

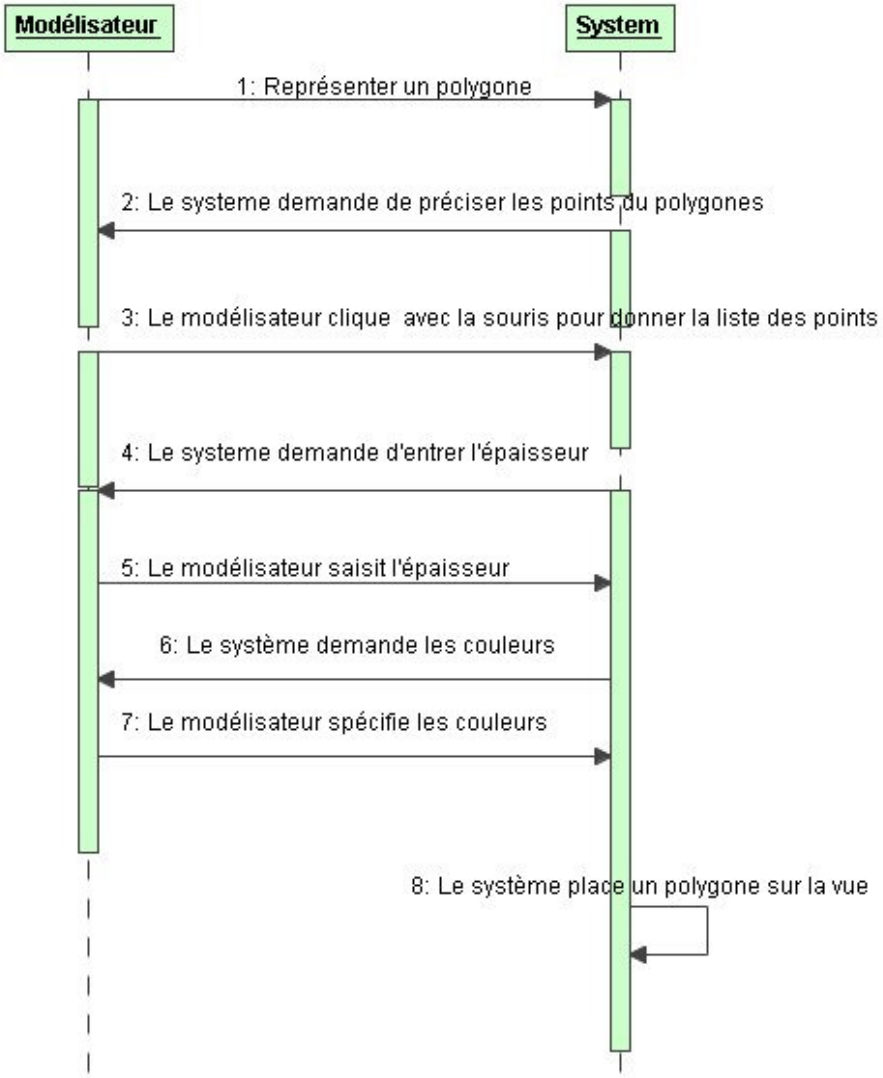


Figure 13: Cas d'utilisation : Représenter un polygone, Diagramme de séquence du scénario nominal

❖ **Représenter ligne brisée**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton dessiner ligne brisée.

Pré condition : Une vue est déjà chargée.

Post condition : Une ligne brisée est positionnée sur la vue.

Fin : Le cas d'utilisation se termine lorsque le système dessine un polygone sur la vue

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur demande de dessiner une ligne brisée.	**2**. Le système demande d'entrer les points composant la ligne brisée
3. Le modélisateur donne la liste des points.	5. Le système demande d'entrer les couleurs.
5. Le modélisateur spécifie les couleurs.	6. Le système demande d'entrer l'épaisseur
7. Le modélisateur spécifie l'épaisseur.	8. Le système dessine une ligne brisée dans la vue.

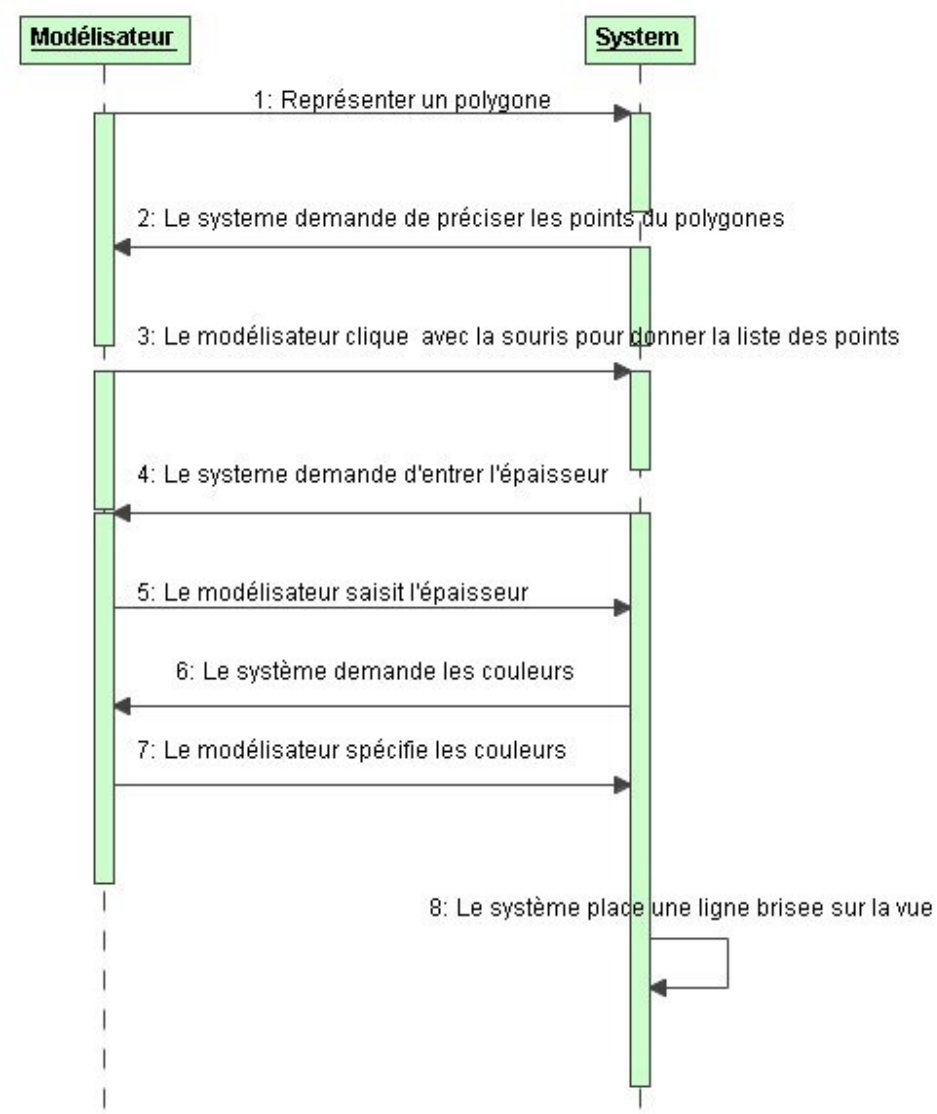


Figure 14: Cas d'utilisation : Représenter une ligne brisée, Diagramme de séquence du scénario nominal

❖ **Créer un objet complexe**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton nouvel objet.

Fin : Le cas d'utilisation se termine lorsque le système crée un nouvel objet sur la vue

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur choisit de créer un objet complexe.	
	2. Le système demande de donner les formes de base constituant la forme complexe.
3. Le modélisateur donne les formes de base ;	
	4. Le système affiche les formes de base au fur et à mesure.
	5. Le système affiche la forme complexe.
	6. Le système demande de sauvegarder la forme complexe.
7. Le modélisateur donne le chemin d'accès	8. Le système sauvegarde la forme complexe.

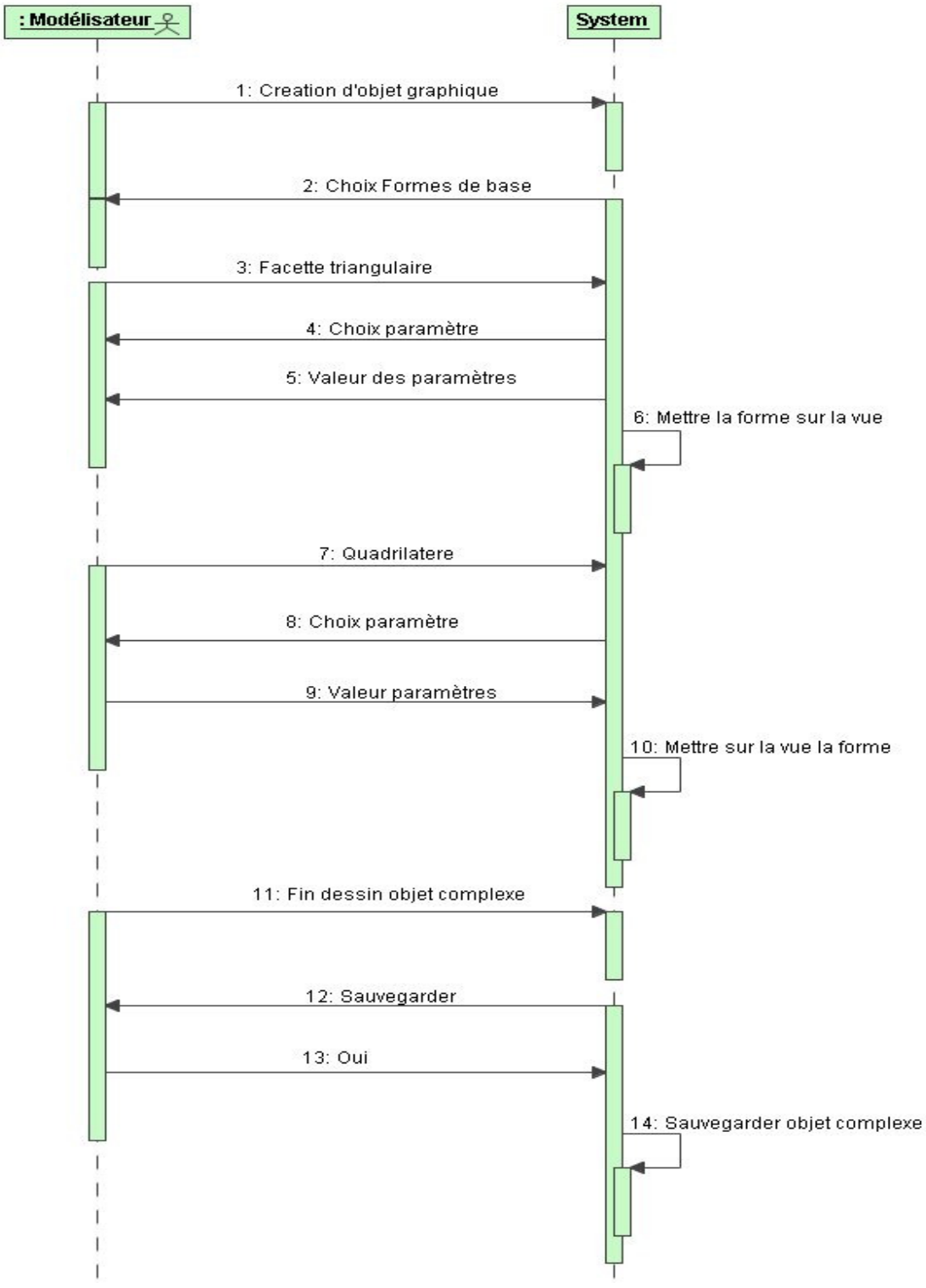


Figure 15: Cas d'utilisation : Créer objet complexe, Diagramme de séquence du scénario nominal

Enchaînement alternatif :

Enchaînement au point 4 :

La forme de base choisie ne partage rien avec celle existant dans la forme complexe.

Echec de dessin.

Reprise au point 4.

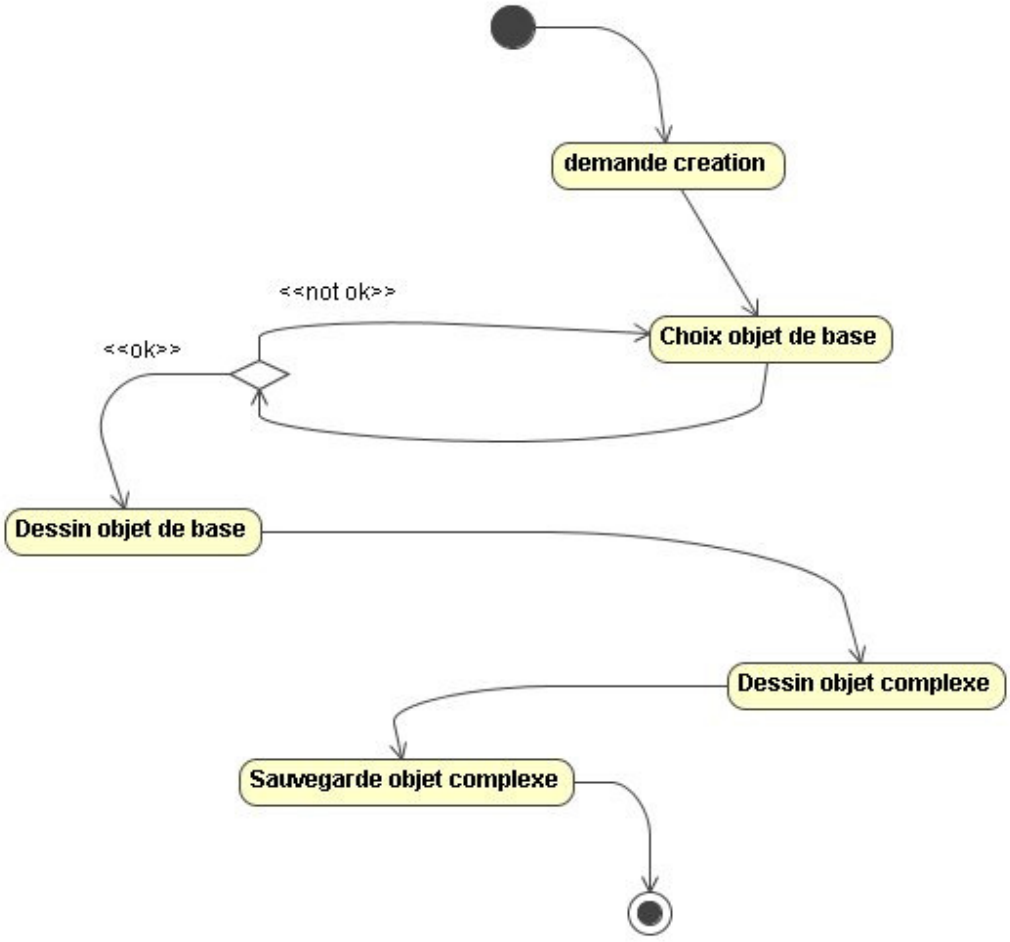


Figure 16: Cas d'utilisation : Créer objet complexe, Diagramme d'activité

❖ **Placer une image**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton nouvel objet.

Fin : Le cas d'utilisation se termine lorsque le système crée un nouvel objet sur la vue

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur choisit d'ajouter une image à la vue.	
	2. Le système demande de spécifier le chemin d'accès du fichier image.
2. Le modélisateur spécifie le chemin d'accès du fichier à insérer.	
	3. Le système demande la position de l'image
5. Le modélisateur saisit les coordonnées du centre.	
	6. Le système affiche l'image à la position spécifier.

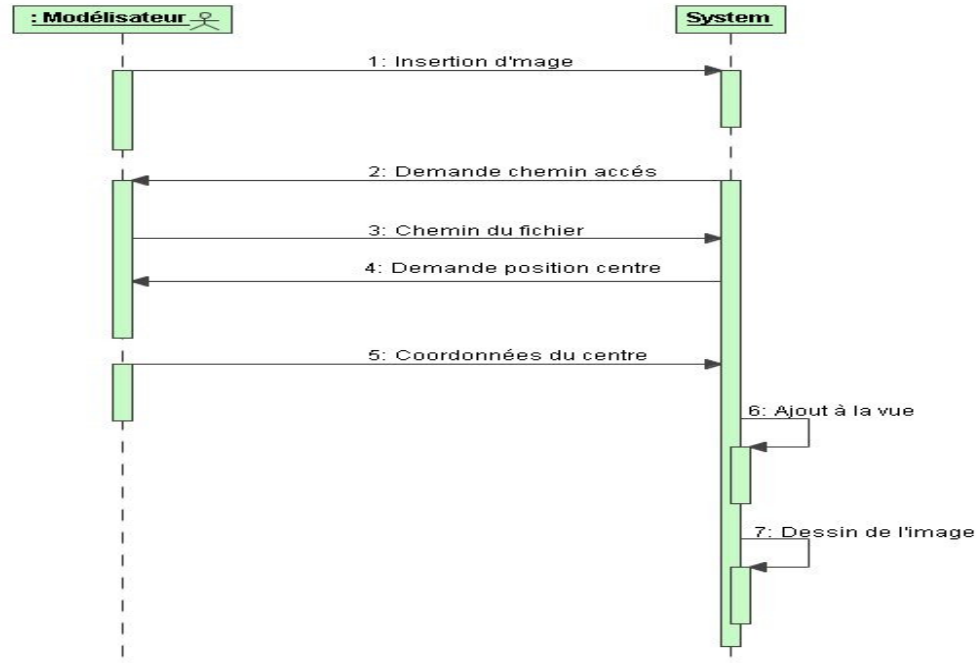


Figure 17: Cas d'utilisation : Placer image: Diagramme de séquence

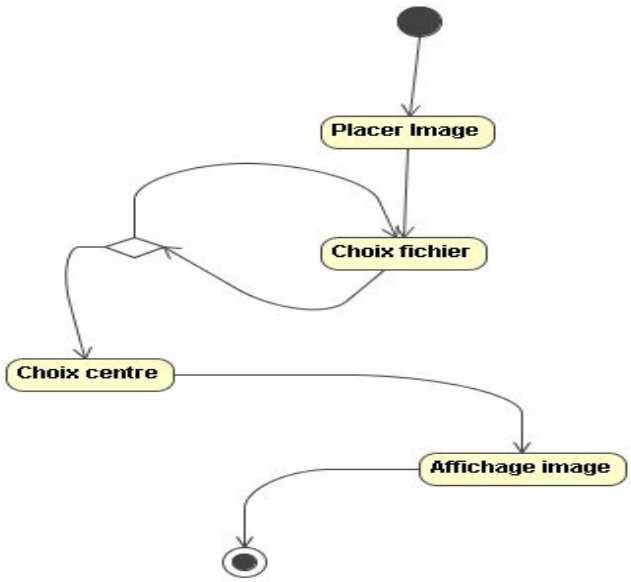


Figure 18: Cas d'utilisation : Placer image: Diagramme d'activité

❖ **Lancer simulation:**

Acteur principal : Modélisateur

Début : Le cas d'utilisation commence lorsque le modélisateur clique sur le bouton Lancer la simulation.

Pré condition : Une vue est déjà chargée ; un modèle MIMOSA est associé à la vue.

Fin : Le cas d'utilisation se termine lorsque le modélisateur clique sur le bouton arrêter ou la durée totale spécifiée est atteinte.

Scénario nominal :

Modélisateur	**Système**
1. Le modélisateur clique sur le bouton « Lancer la Simulation ».	
	2. Le Système demande d'entrer la durée d'une étape (step).
3. Le modélisateur saisit la durée d'une étape.	
	4. Le Système demande d'entrer la durée totale de la simulation.
5. Le modélisateur saisit la durée totale de la simulation.	
	6. Le système réaffiche la vue chaque pas de temps.

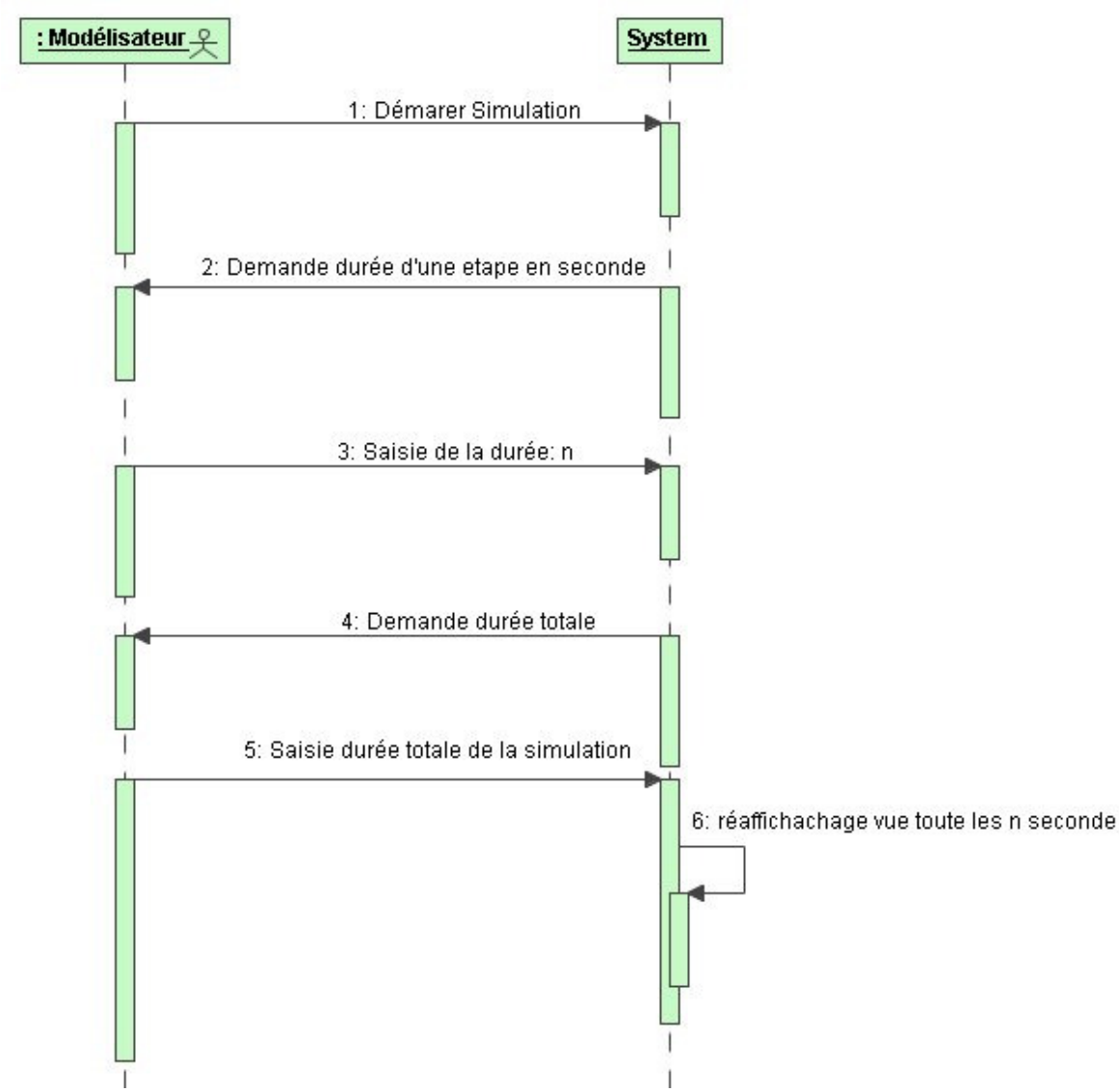


Figure 19: Cas d'utilisation Lancer la simulation: Diagramme de séquence

3. Vues statiques du système

En ce qui concerne la visualisation de modèle MIMOSA nous avons besoin des classes nous permettant de définir des objets géométriques de base de sorte que tout modèle MIMOSA puisse être visualisé. Pour cela nous avons jugé nécessaire d'avoir les classes suivantes :

***Eléments ponctuels* :**

Point, Icône, Texte2D, Chernoff Face

- ***Eléments linéaires*** *:*

  Ligne Brisée

- ***Eléments surfaciques*** *:*

  Polygone, Rond

- ***Eléments volumiques*** *:*

  Parallélépipède, Cône, Cylindre, Sphère, Forme Complexe 3D, Texte3D

A ces classes, il faudra ajouter celles permettant de gérer des regroupements d'objets géométriques.

MultiPoint, MultiIcône, MultiChernoffFace, MultiLigneBrisée, MultiArc, MultiRond, MultiPolygone, MultiTexte2D, MultiTexte3D, Plan (Zone de dessin), Collection

Le diagramme des classes :

Les diagrammes de classes constituent le noyau du langage *UML*. Ces diagrammes offrent une vue statique du système, en représentant les classes et les relations entre les classes.

Un diagramme de classe UML représente un ensemble d'éléments déclaratifs (statique) du modèle comme les classes, les types ainsi que leurs contenus et relations.

Une classe est une construction standard d'UML utilisée pour spécifier le canevas à partir duquel les objets seront fabriqués à l'exécution. Une classe est une spécification, un objet est une instance d'une classe. Les classes peuvent hériter d'autres classes (c'est-à-dire qu'elles peuvent hériter de tous les comportements et attributs de leurs parents et rajouter des nouvelles fonctionnalités qui leurs sont propres), avoir des attributs ayant d'autres classes comme type, déléguer des responsabilités à d'autres classes et implémenter des interfaces abstraites. Le Modèle de Classe est au cœur de la conception et du développement orienté objet. Il exprime aussi bien l'état statique que le comportement du système. Une classe encapsule l'état (attributs) et fournit des services pour manipuler cet état (comportement). Une bonne

conception orientée objet limite les accès directs aux attributs et offre des services permettant de manipuler les attributs pour le compte de l'appelant. Ce masquage des données et la fourniture des services assure que les données soient mises à jour dans un seul endroit et suivant des règles bien spécifiques – la maintenance du code des grands systèmes ayant été programmés avec des accès directs à un élément de données dans plusieurs endroits étant un grand fardeau.

Quelques relations pouvant exister entre des classes du modèle :

Association : Cette relation est représentée par un trait plein, pouvant être orienté. La multiplicité est notée du côté du rôle cible de l'association. Elle spécifie le nombre d'instances pouvant être associées (liées) avec une seule instance source de la relation.

Héritage : Relation d'héritage, dans laquelle les objets de l'élément spécialisé (classe enfant) peuvent remplacer les objets de l'élément général (classe parent).

Les classes ci-dessus peuvent être structurées autour des diagrammes de classes suivants :

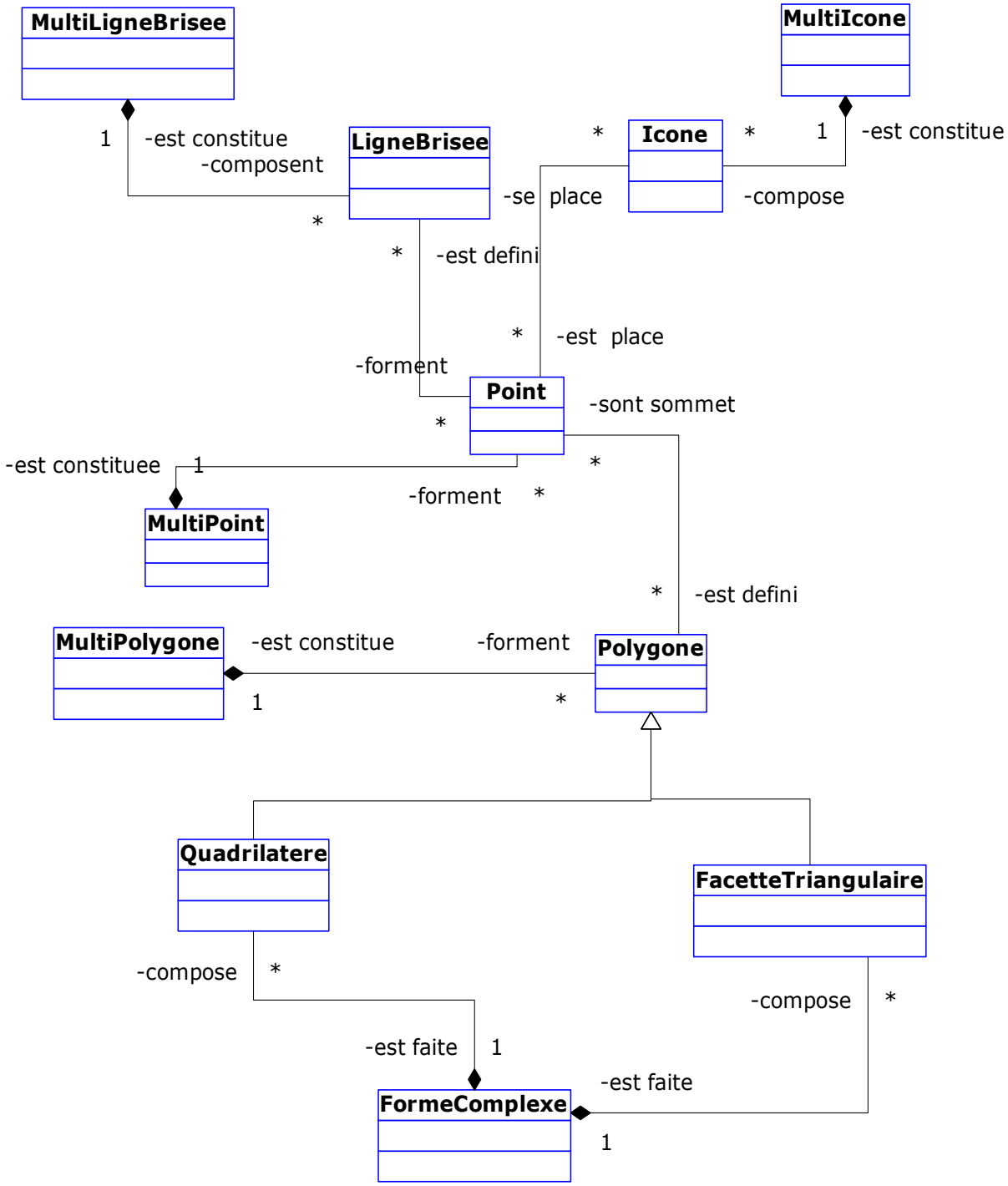


Figure 20: partie1 du diagramme de classes

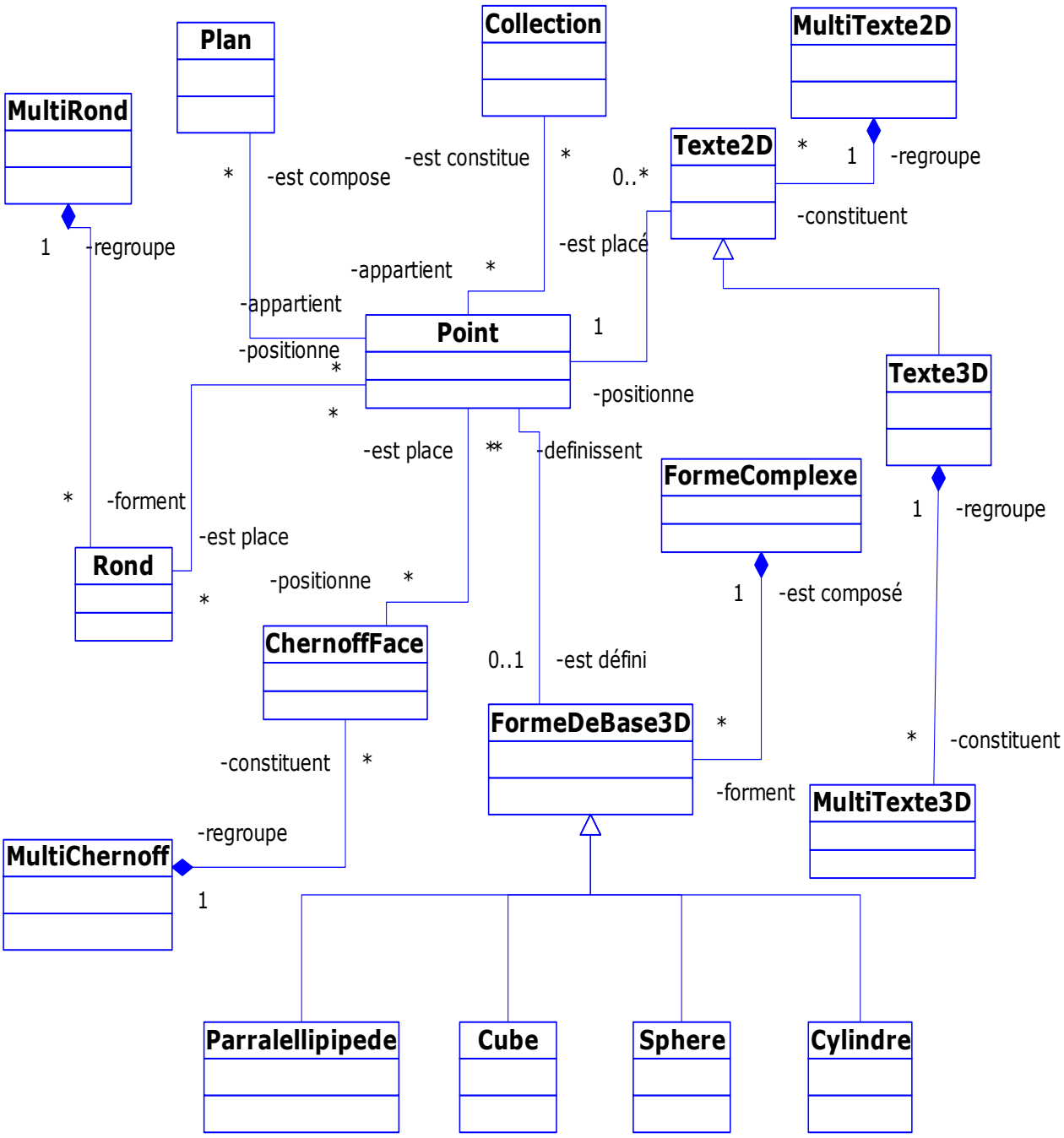


Figure 21: partie2 du diagramme de classes du système

DESCRIPTION DE QUELQUES CLASSES *:*

Classes	***Paramètres variables***	***Evénements***
Point	- Abscisse - Ordonnée - Cote - Epaisseur - Couleur	- Déplacer - Colorer - Modifier Epaisseur
Icône	- Taille - Position - Chemin d'accès	- Déplacer - Modifier Taille - Modifier chemin
Texte2D	- Police - Taille - Couleur - Valeur - Position	- Déplacer - Modifier Taille - Colorer - Modifier Chaîne
Chernoff Face	- Position - Couleur - Liste Faces	- Déplacer - Colorer -
Ligne Brisée	- Liste Points (ordonnée) - Couleur - Epaisseur	- Déplacer - Colorer - Modifier Epaisseur
Polygone	- Couleur de la ligne - Epaisseur de la ligne - Couleur de l'intérieur - Liste ordonnée des sommets	- Colorer la ligne - Colorer l'intérieur - Modifier l'épaisseur de la ligne - Déplacer : (transformations géométriques).

Rond *(Nous avons là les cercles et les ellipses)*	- Couleur de la ligne - Epaisseur de la ligne - La plus grande distance suivant l'horizontale (D) - La plus grande distance suivant la verticale (d)	- Modifier D - Modifier d - Colorer la ligne - Colorer l'intérieur - Modifier l'épaisseur de la ligne - Déplacer
Cône	- Rayon - Hauteur - Position	- Colorer - Modifier rayon - Modifier hauteur - Déplacer
Cylindre	- Rayon - Hauteur - Position - Couleur	- Colorer - Modifier rayon - Modifier hauteur - Déplacer
Sphère	- Rayon - Position - Couleur	- Colorer - Modifier rayon - Déplacer
Parallélépipède	- Longueur - Largeur - Hauteur - Position - Couleur	- Modifier longueur - Modifier largeur - Modifier hauteur - Colorer - déplacer
Forme Complexe	- Liste des sommets - Couleur des sommets	- Déplacer -
Texte3D	- Police - Taille - Couleur - Valeur - Position	- Déplacer - Modifier Taille - Colorer - Modifier Chaîne

Plan	- Image de fond - Longueur - Largeur - Liste des composants	-
Collection *(Canvas3D)*	- Eclairage - Camera - Liste des composants	-

4. Intégration au logiciel MIMOSA

La visualisation de l'évolution d'un modèle MIMOSA est prise en compte au niveau de la couche 2. En effet cette couche met à disposition les métas concepts pour décrire des modèles. La vue sera considérée comme un modèle MIMOSA.

L'intégration dans MIMOSA consistera à mettre à disposition les métas concepts permettant de définir des vues dans la plate-forme MIMOSA. Toutefois, il faut prendre en compte le fait que MIMOSA est construit sur la base d'une couche introduisant les notions de composant, composé, relation ; et que tout modèle MIMOSA autrement dit tout discours dérive de ces métas concepts qui constituent le méta discours.

La vue étant considérée comme un composé, les objets, la constituant, sont considérés comme ses composants. Ce qui donne le diagramme de classes suivant qui matérialise l'intégration des éléments visuels c'est-à-dire la vue et ses composants aux éléments de base de MIMOSA, composant-composé-relation.

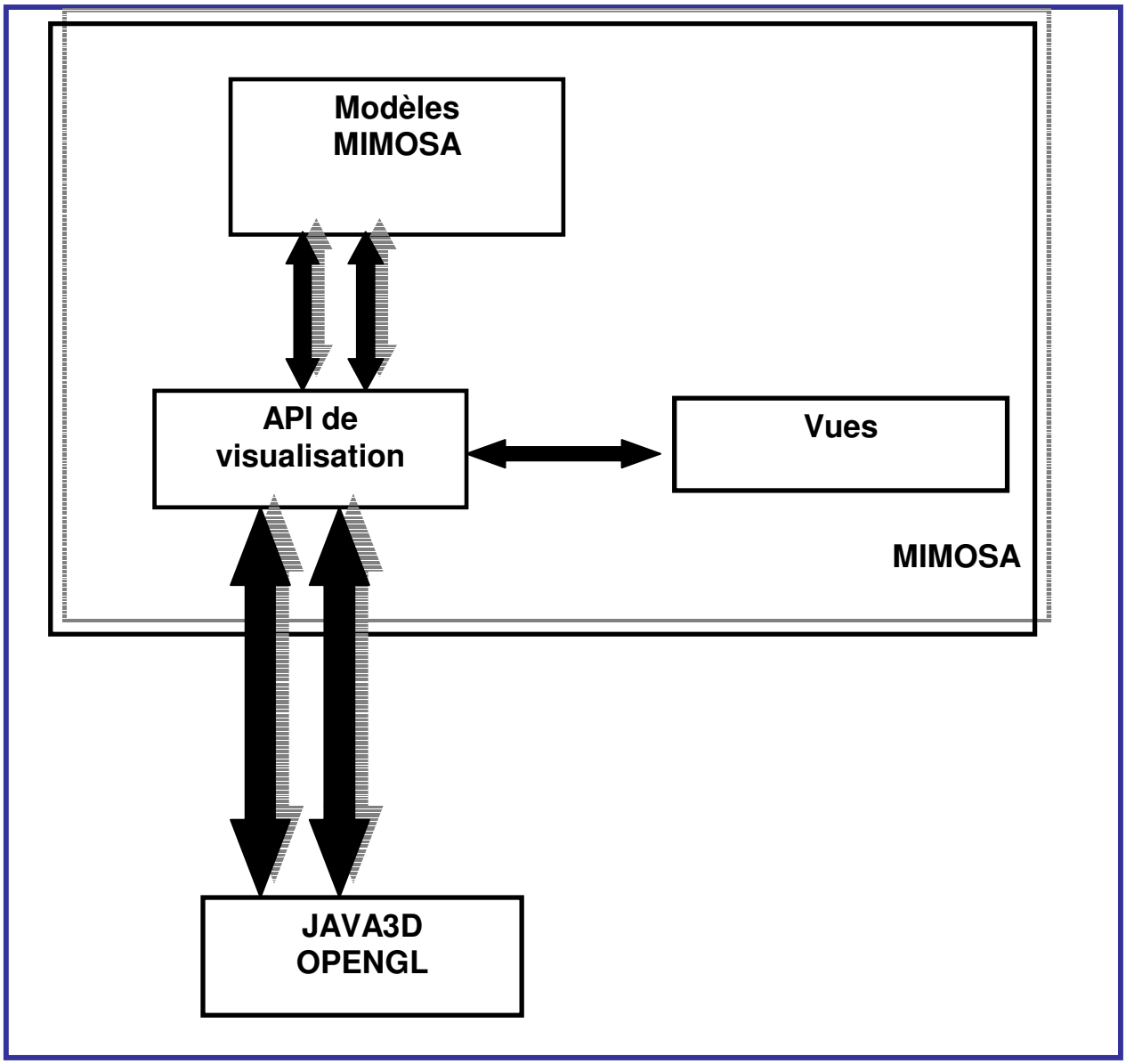


Figure 22: Architecture du système

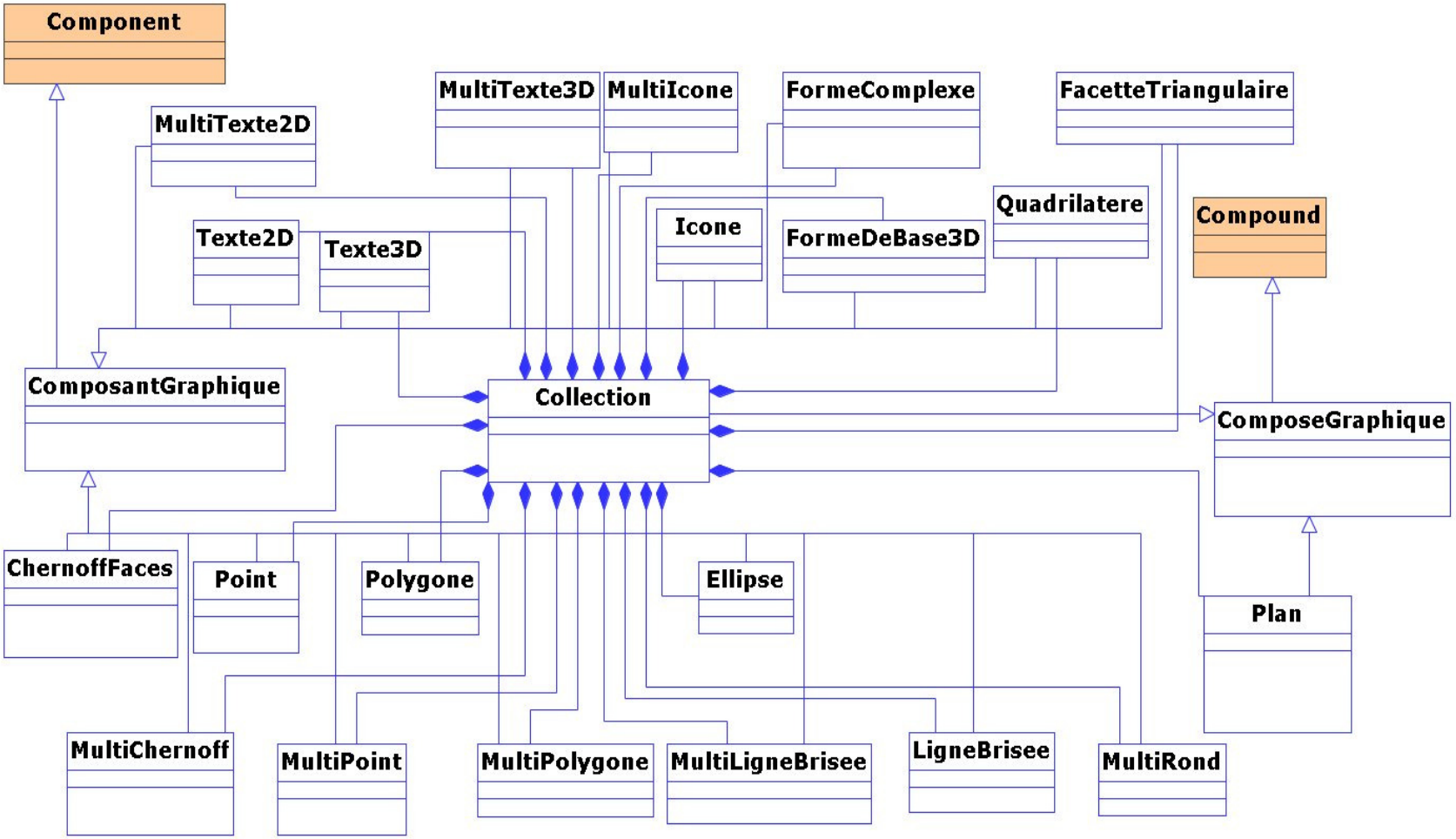


Figure 23: Diagramme de classe Intégration dans MIMOSA.

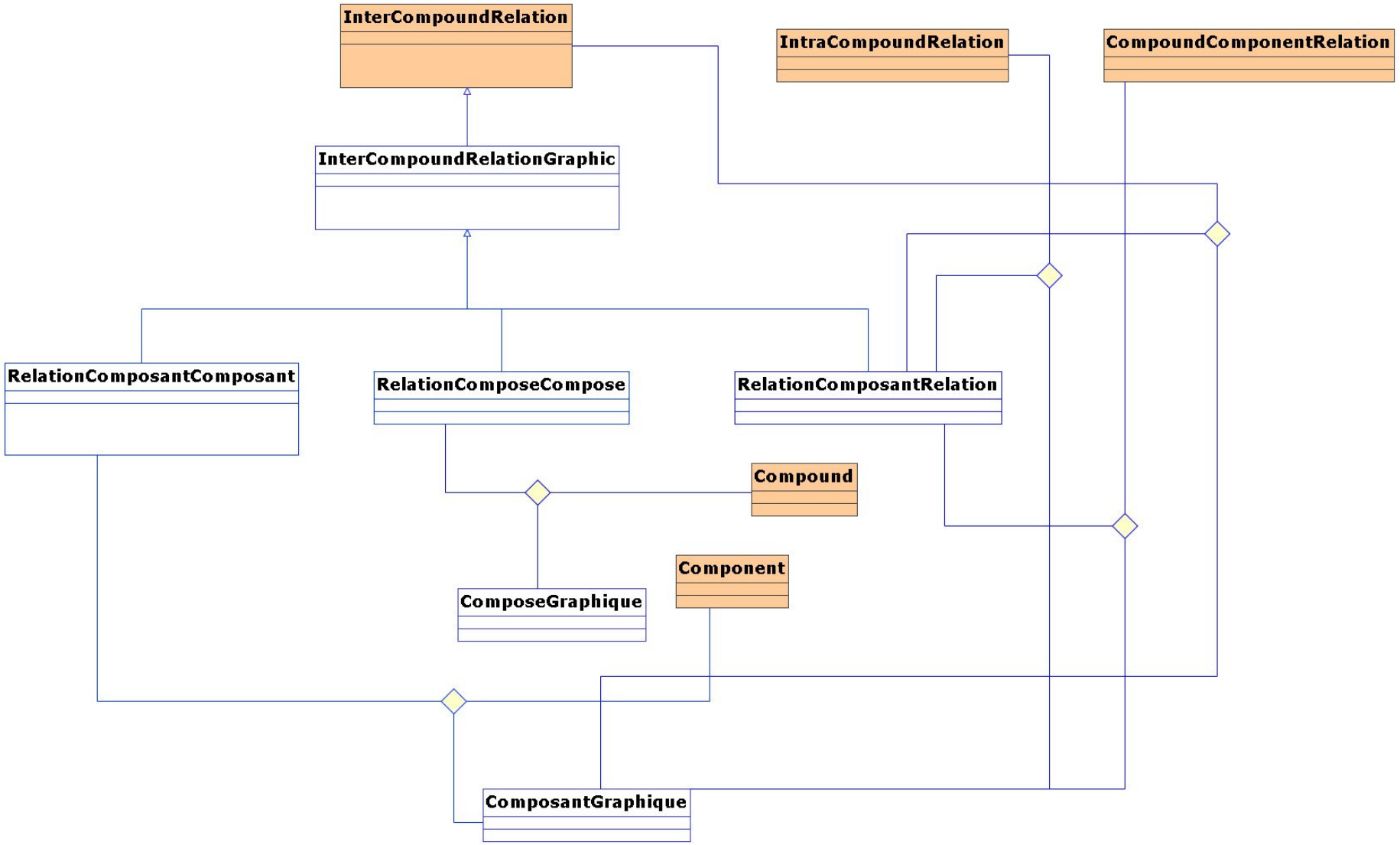


Figure 24: Diagramme de classes: intégration dans MIMOSA

5. Explication détaillée du modèle proposé

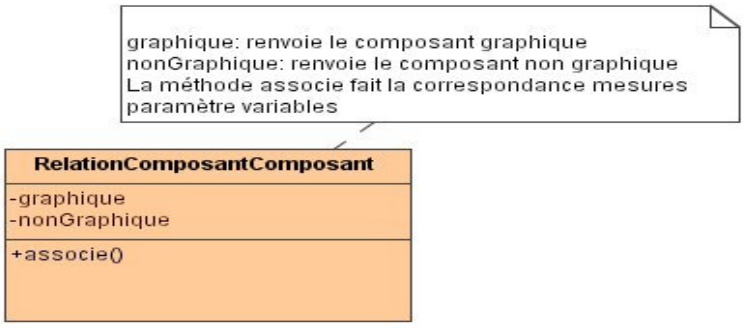
Le modèle proposé est composé d'un ensemble de classes reliées par des relations (InterCompoundRelationType au sens MIMOSA). Ces classes sont les composants de la vue (qui peut être un Plan ou un Canvas3D selon que l'on travaille en 3D ou en 2D). De ce fait, nous définissons les seuls composés Plan et Canvas3D.

Les relations qui sont les plus importants sont les liens entre la vue et le modèle MIMOSA. De ce fait nous distinguons :

- La relation entre un composant graphique de la vue et un composant du modèle MIMOSA. Cette relation informe sur les correspondances entre mesures du composant du modèle et paramètres variables du composant graphique.
- La relation entre une relation du modèle MIMOSA et un ensemble de composants graphiques.
- Nous avons également la relation triviale entre le modèle et la vue.

Nous avons besoin de deux classes essentiellement :

- *RelationComposantComposant :*



- *RelationComposantsRelation*

Exemple :

Nous avons :

- un automate cellulaire (un composé fait de cellules avec une relation d'adjacence)
- une population d'agent (un composé fait d'agents)
- une vue permettant d'afficher
    - les cellules comme des carrés dont la surface (et/ou) l'épaisseur du contour varie en fonction des valeurs de la cellule
    - les agents sous forme de points ou d'icône dont la couleur et la forme varient en fonction de l'état des agents
- une relation entre l'automate et la vue et une autre entre la population d'agent et la vue de telle sorte à voir les agents se déplacer sur la grille de l'automate cellulaire

Solution *:*

Pour la résolution de ce problème, nous pouvons considérer que :

- *La vue est un Canvas2D composé de :*
    - *Un MultiPolygone*
    - *Un MultiPoint*
- *L'automate cellulaire est associé à un MultiPolygone (objet graphique)*
- *Les cellules sont des Polygone : Relation entre le composant cellule et le composant graphique Polygone.*

  La cellule ayant une valeur (mesure), cette dernière sera associé à un paramètre variable du composant graphique Polygone (Par exemple la couleur de la surface, on pourra dire que si la valeur est inférieur à x la couleur sera jaune,…).
- *La population d'agents correspond à un MultiPoint.*
- *Les agents correspondent à des Point. Le paramètre position du Point sera associé à la position de l'agent concerné.*

  Le déplacement des agents se fera par un changement de position. Les couleurs des Point pourront correspondre aux états des agents.

Pour ce faire, il faudra instancier deux objets RelationComposantComposant :

- *Un pour relier les agents aux points*
- *Un autre pour relier les cellules aux polygones*

CHAPITRE 4 : IMPLEMENTATION

1. Outils mathématiques

Le graphisme a toujours posé un réel problème concernant la rapidité. En effet le graphisme utilise de lourds calculs mathématiques. L'un des objectifs des grandes compagnies informatiques (SUN, MICROSOFT,…) était alors de trouver les voies et moyens pour alléger les calculs. Ces concepteurs de logiciels ont été aidés par les constructeurs de matériels dans cette démarche. En effet ces derniers ont mis en place de puissantes cartes graphiques dotées d'une mémoire. Ce qui allège le travail du processeur.

Faire du graphisme en informatique signifie tout simplement faire de la vision par ordinateur. A cet effet il faut impérativement un espace où visualiser les éléments graphiques. Ceci n'est rien d'autre que l'écran. Il s'agira de définir un repère au niveau de l'écran et d'y positionner les objets géométriques (les formes).

L'écran est cependant très petit pour visualiser tout ce que l'on voudrait. En ne considérant que l'écran sans l'application des mathématiques il serait impossible de visualiser la carte du SENEGAL. Dès lors, on s'aperçoit de l'utilité des mathématiques. En effet, on se servira des calculs numériques pour effectuer les transformations nécessaires et représenter ainsi ce que l'on voudrait sur écran d'ordinateur.

Pour faire du graphisme en informatique, nous sommes obligés d'utiliser certains calculs propres aux mathématiques.

Nous introduisons à ce niveau la notion de coordonnées homogène qui se trouve idéale en temps de calcul.

1.1. La translation

La translation est une transformation géométrique qui simule un déplacement suivant une direction.

La translation est une application :

$T : IR^n : \longrightarrow IR^n$

$X \longrightarrow Tr+X$

Où Tr est un vecteur

En 3D: Si on a X' = T(X) avec X= $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ et T = $\begin{pmatrix} Tx \\ Ty \\ Tz \end{pmatrix}$

Alors :

X' = $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ avec $\begin{cases} x' = Tx + x \\ y' = Ty + y \\ z' = Tz + z \end{cases}$

1.2. La rotation

La rotation est une transformation géométrique qui simule un déplacement autour d'un axe, d'un angle α donné.

La rotation est une application :

R: IR^n : IR^n ⟶

X RX ⟶

Où R est une matrice de la forme : (en coordonnées homogènes)

Suivant l'axe des x : $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Suivant l'axe des y : $\begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Suivant l'axe des z : $\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

La rotation par rapport à un axe différent de ces trois axes de base n'est rien d'autre qu'une combinaison de ces rotations.

1.3 Le changement d'échelle

Comme les deux applications ci-dessus, le changement d'échelle est une multiplication par une matrice donnée.

La matrice est de la forme : (en coordonnées homogènes)

$$E = \begin{bmatrix} e_x & 0 & 0 & 0 \\ 0 & e_y & 0 & 0 \\ 0 & 0 & e_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ces différents outils permettent d'écrire des algorithmes simples pour gérer un monde tridimensionnel.

2. Choix de l'outil graphique : le moteur OPENGL

Il a été développé en 1989 (GL) par Sillicon Graphics, puis portée sur d'autres architectures en 1993 (OpenGL).

OpenGL est une librairie graphique 3D. Cela signifie qu'on lui donne des ordres de tracé de primitives graphiques (facettes, etc.) directement en 3D, une position de caméra, des lumières, des textures à plaquer sur les surfaces, etc., et qu'à partir de là, OpenGL se charge de faire les changements de repère, la projection en perspective à l'écran, l'élimination des parties cachées, d'interpoler les couleurs, et de *rasteriser* (tracer ligne à ligne) les faces pour en faire des pixels.

OpenGL s'appuie sur le hardware disponible selon la carte graphique. Toutes les opérations de base sont a priori accessibles sur toute machine, simplement elles iront plus ou moins vite selon qu'elles sont implémentées en hard ou pas.

La machine OpenGL (*graphic engine*) se décompose en plusieurs moteurs:

- Le *geometric engine* s'occupe de la partie purement 3D: changements de repère, projection en perspective à l'écran.
- *le raster engine* prend en charge la partie 2D: il rasterise les triangles de manière à produire des pixels, interpole au passage les couleurs et les coordonnées texture,

puis élimine les parties cachées en fonction de la profondeur z du fragment en cours de tracé et de la profondeur actuellement stocké dans le pixel visé. Ceci constitue le schéma de base. OpenGL offre de nombreux mécanismes supplémentaires et moyens de contrôle.

- *le raster manager*, à qui le pixel est confié pour être tracé à l'écran après d'éventuels traitements supplémentaires.

Nous avons à coté de OpenGL, l'API Direct3D développée par Microsoft. Direct3D étant une API propriétaire et spécifique à WINDOWS, nous sommes rapidement bloqué dans son utilisation devant OpenGL qui est multi plate-forme, et encore très puissant.

NB : Nous avons donc choisi d'utiliser la librairie OPENGL

JAVA de son coté offre une librairie de classes (JAVA3D) qui jouent le un rôle d'interface avec OPENGL.

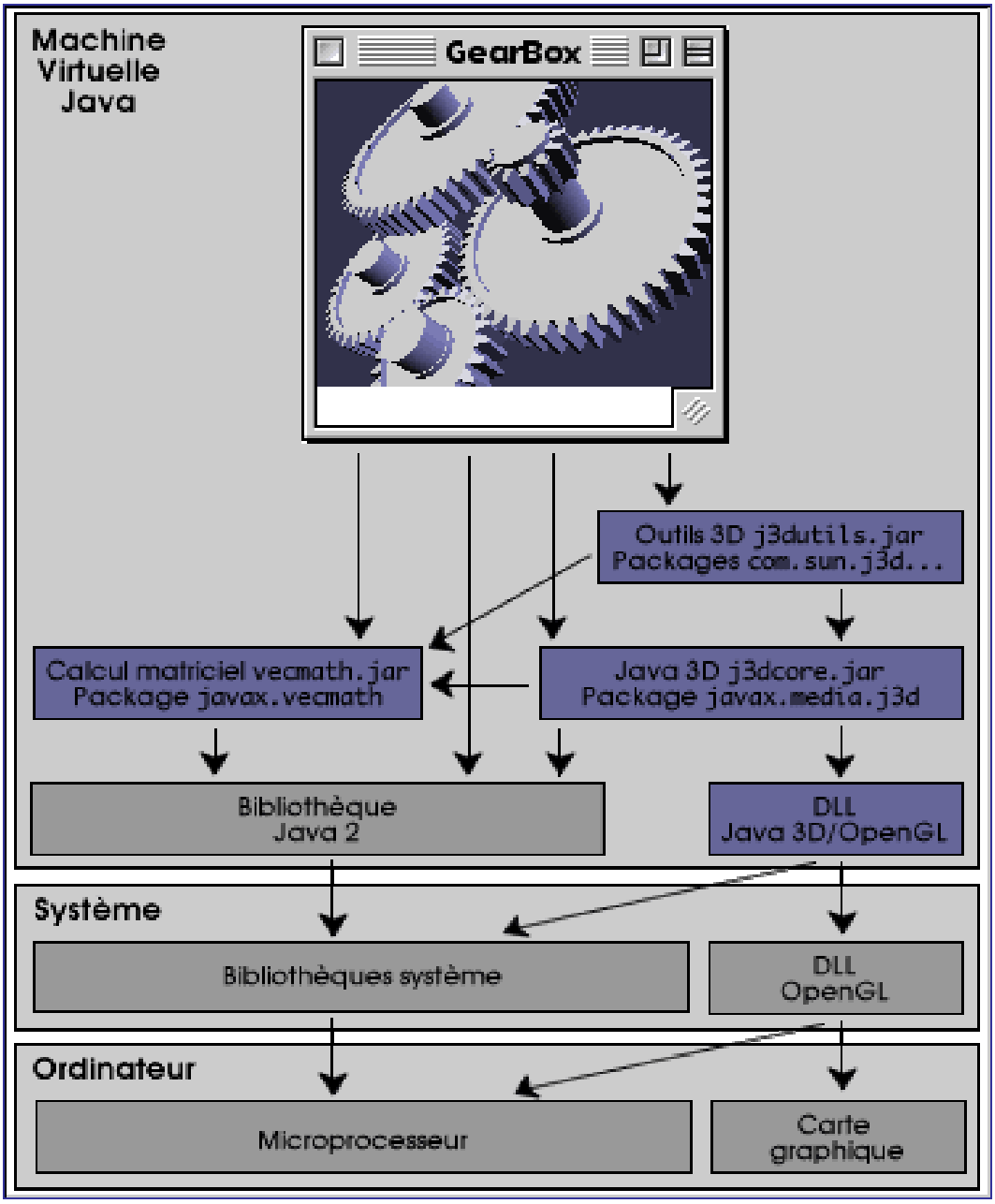


Figure 25: Architecture OPENGL/JAVA3D

3. Stockage des données

Comme dans toute application informatique, le stockage de données est un maillon très important dans le système de visualisation de l'évolution de modèle MIMOSA. En effet, il est

indispensable de pouvoir sauvegarder certaines données. Par exemple lors de la sauvegarde d'une vue il faudra bien sûr stocker les données quelque part.

Nous avons des systèmes de gestion de bases de données capable de faire le travail mais également on peut se servir de XML pour réaliser un tel objectif.

XML et les SGBD

Nous assistons par les temps qui courent à une utilisation de fichiers XML à la place des fichiers simples (non structurés) par les SGBD. Parmi ceux qui offrent cette technologie nous pouvons citer SQL Server 2000 - SQL XML, ORACLE 9i. Le choix se trouve donc entre le stockage aussi bien de la structure et des informations dans des fichiers XML et l'utilisation d'un SGBD qui ne servirait que d'interface dans certains cas.

Notons cependant que l'utilisation d'un SGBD, rendrait MIMOSA dépendant de ce dernier.

Ce qui conduit à utiliser XML.

Conclusion

Le projet MIMOSA est très ambitieux et sa réalisation s'étale sur plusieurs années. L'étude ci-dessus nous a permis d'avoir une vue assez large sur la simulation des systèmes complexes. Par rapport aux résultats attendus, on peut dire que la mise en place de MIMOSA n'est pas encore terminée. Toutefois, notons que la partie qui nous été confiée a été développée (mise à part quelques imperfections à corriger). L'API est en cours de test sur un modèle précis (les robots mineurs). Par ailleurs, c'est à partir de ces tests qui sont des expériences déjà réalisées sur d'autres plates-formes, que se montrera la pertinence de MIMOSA.

L'étude de la visualisation de l'évolution de modèle MIMOSA est passée non seulement par un état de l'art sur les systèmes de visualisation de certaines plates-formes existantes mais aussi par une utilisation des outils graphiques des langages de programmation notamment JAVA et la librairie JAVA3D (qui ont servi à mettre en place l'API).

La prise en compte complète des systèmes d'information géographiques faisait partie des objectifs du système de visualisation mais auparavant, il faudrait une extension de MIMOSA (c'est-à-dire construire le discours sur les SIG). De même, il est possible d'utiliser les classes servant à la visualisation pour l'édition de modèles, encore faudrait-il construire un discours sur l'édition de modèles. Ces deux extensions restent dans la partie définition des API génériques nécessaires aux modélisateurs. Il reste donc des choses à faire et à améliorer. Toutefois, il serait intéressant d'étendre MIMOSA de sorte qu'il puisse prendre en compte directement des modèles réalisés sous d'autres plates-formes (MADKIT,…). La question qu'il faut se poser, à partir de ce moment est de savoir : Est ce que MIMOSA sera le standard de la modélisation et de la simulation ?

ANNEXES

ANNEXE 1 : JAVA2D

Java 2D est un ensemble de classes facilitant le dessin 2D avec Java. Java 2D offre une classes graphique (Graphics2D) très puissante permettant de dessiner. La classe Graphic2D implémente des méthodes qui gèrent l'affichage évolué (en dimension 2) avec Java.

Quelques méthodes de la classe Graphics2D : (La liste n'est pas exhaustive)

Méthodes de dessin relatives aux objets vectoriels

- Segments de droites:
  - void drawLine(int x1,int y1,int x2,int y2);
- Rectangles remplis ou non:
  - void drawRect(int x,int y, int width,int height);
  - void fillRect(int x,int y, int width,int height);
- Rectangles avec sommets arrondis remplis ou non:
  - void drawRoundRect(int x,int y, int width,int height, int arcWidth, int arcHeight);
  - void fillRoundRect(int x,int y, int width,int height, int arcWidth, int arcHeight);
- Cercles et ellipses remplis ou non:
  - void drawOval(int x,int y, int width,int height);
  - void fillOval(int x,int y, int width,int height);
- Arcs de cercles et d'ellipses remplis ou non:
  - void drawArc(int x,int y, int width,int height, int startAngle,int arcAngle);
  - void fillArc(int x,int y, int width,int height, int startAngle,int arcAngle);
- Polygones remplis ou non:
  - void drawPolygon(int[] xPoints, int[] yPoints, int nPoints);
  - void drawPolygon(Polygon p);
  - void fillPolygon(int[] xPoints, int[] yPoints, int nPoints);
  - void fillPolygon(Polygon p);

- Formes évoluées remplies ou non:
  - void draw(Shape s);
  - void fill(Shape s);

Méthodes relatives aux attributs graphiques

- Utilisation d'un pinceau d'affichage pour les bords et les intérieurs des objets graphiques affichés:
  - void setStroke(Stroke s);
  - void setPaint(Paint paint);

Méthodes relatives aux images bitmap et la gestion des pixels

- Fonctions d'affichage des images:
  - boolean drawImage(Image img,int x,int y, Color bgcolor, ImageObserver observer);
  - boolean drawImage(Image img,int x,int y, ImageObserver observer);
  - boolean drawImage(Image img,int x,int y, int width,int height, Color bgcolor, ImageObserver observer);
  - boolean drawImage(Image img,int x,int y, int width,int height, ImageObserver observer);
  - boolean drawImage(Image img, int dx1,int dy1, int dx2,int dy2, int sx1,int sy1, int sx2,int sy2, Color bgcolor, ImageObserver observer);
  - boolean drawImage(Image img, int dx1,int dy1, int dx2,int dy2, int sx1,int sy1, int sx2,int sy2, ImageObserver observer);
  - void drawImage(BufferedImage img, BufferedImageOp op, int x, int y);
  - boolean drawImage(Image img, AffineTransform xform, imageObserver obs);

Méthodes relative au texte et aux polices de caractères

Affichage de texte

- void drawString(AttributedCharacterIterator ite, float x,float y);
- void drawString(AttributedCharacterIterator ite, int x,int y);
- void drawString(String s, float x,float y);

- void drawString(String str,int x,int y);
- void setFont(Font font);

En outre nous avons un certain nombre de classes concernant les formes géométriques. Toutefois la classe Graphics2D gère une grande partie du dessin en 2D.

ANNEXE 2 : JAVA3D

Java3D est une API regroupant un ensemble de classes permettant de définir et de visualiser un monde 3D. L'API Java3D permet en outre de gérer l'accélération matérielle. L'API Java3D est basée sur un modèle de graphes pour la représentation d'une scène. Le graphe décrit entièrement la scène (monde virtuel) :

- Les données géométriques,
- les attributs pour le rendu,
- les points de vues.

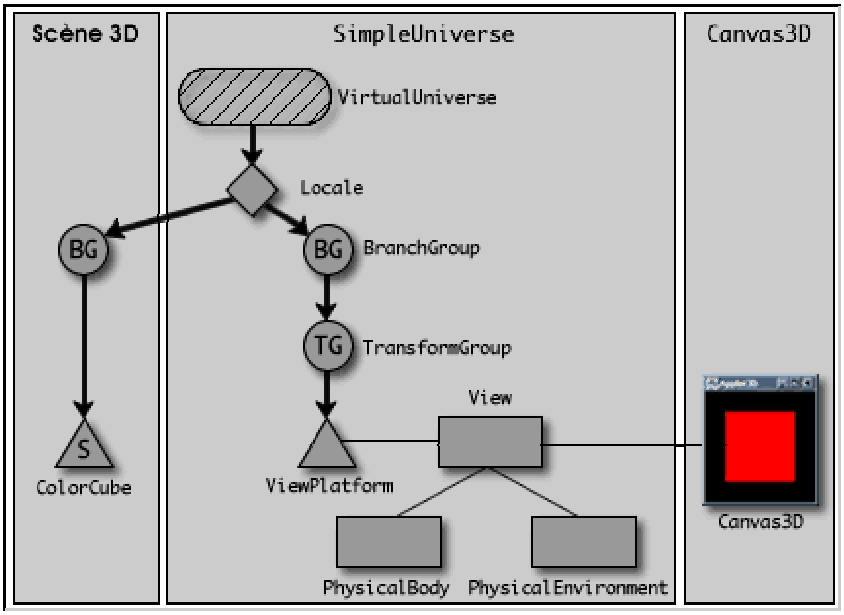
Les classes de l'API Java 3D appartiennent au package javax.media.j3d. Le package javax.vecmath (fichier vecmath.jar) rassemble les classes dédiées au calcul vectoriel et matriciel dont Java 3D a besoin. Bien que les classes des packages javax.media.j3d et javax.vecmath soient suffisantes pour programmer toutes les fonctionnalités offertes par Java 3D, des classes utilitaires facilitant la programmation sont fournies dans les packages commençant par com.sun.j3d... (Fichiers j3dutils.jar et j3daudio.jar).

Concernant les formes géométriques le package le plus important est javax.media.j3d.Geometry.

Principes 3D

Construction d'un univers 3D

De quoi a-t-on besoin ?



Composants d'un univers 3D

- *Une instance de Canvas3D* : Ce composant AWT qui dérive de la classe java.awt.Canvas, est la zone de l'écran qui permet de visualiser une scène 3D. Cette zone peut être vue comme la pellicule d'un appareil photo : elle ne permet pas de capturer tous les objets autour de l'appareil mais juste d'avoir une vue plus ou moins large sur un ensemble d'objets en 3 dimensions.

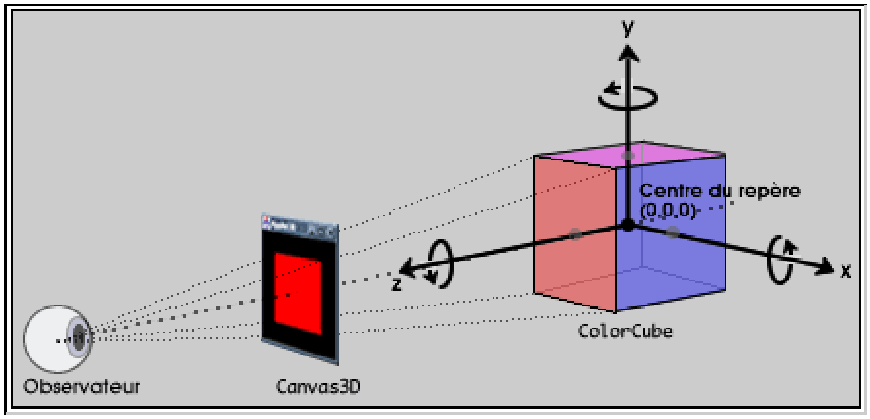
- *Une scène 3D à visualiser* : cette scène est un ensemble d'une ou plusieurs formes géométriques simples (parallélépipède, sphère, cylindre ou cône) ou complexes (ensemble de points reliés entre eux, texte 3D). Sur chacune de ces formes 3D, on peut effectuer des transformations (translation, rotation, homothétie) permettant de les positionner dans l'espace et de les animer.
  L'ensemble de ces formes et de ces transformations est représenté par un graphe de type arbre dont la racine est une instance de BranchGroup.

- *Une instance de SimpleUniverse* : Cet objet permet de relier l'instance de Canvas3D avec la scène 3D à visualiser. Cet objet modélise l'espace dans lequel on positionne l'appareil photo. A cet espace sont rattachés une instance de Canvas3D et une scène 3D. La classe SimpleUniverse est une classe utilitaire qui permet de créer des instances par défaut des

classes VirtualUniverse, Locale, ViewPlatform, View, PhysicalBody et PhysicalEnvironment, nécessaires à Java 3D.

Repère 3D

Une scène 3D est composée d'un ensemble de formes 3D. Chacune de ces formes est construite par un assemblage de triangles ou de quadrilatères plans. Ces formes simples sont décrites grâce aux coordonnées (x, y, z) de leurs sommets. Pour s'orienter correctement dans la scène 3D à laquelle vous allez ajouter des formes 3D, il faut donc connaître comment est positionné le repère 3D sous-jacent à tout espace 3D.



Orientation d'un repère 3D

Transformation 3D

Les transformations sont utilisées en 3D pour positionner et animer une forme 3D dans l'espace. Java 3D utilise la classe Transform3D pour décrire une opération de translation, de rotation ou d'homothétie (changement d'échelle). Cette opération est associée à une instance de la classe TransformGroup et ajoutée à l'arbre de la scène pour l'appliquer sur une forme.

Arbre d'une scène 3D

Un arbre représentant une scène 3D est constitué de noeuds qui sont soit des feuilles, soit des groupes. Les noeuds reliés entre eux ont une relation parent enfant. Tous les noeuds ont un parent sauf la racine et peuvent avoir un ou plusieurs enfants sauf les feuilles.

- Un arbre est un graphe ayant des propriétés particulières : A partir de sa racine, il est possible d'énumérer toutes les feuilles et il existe un chemin unique reliant la racine à chacune des feuilles.

- En Java 3D, une feuille représentant une forme géométrique est un assemblage de triangles ou de quadrilatères décrits par les coordonnées (x, y, z) de leurs sommets.

- Le chemin qui relie la racine à une feuille est un ensemble de groupes. Java 3D part de chaque feuille puis remonte le chemin jusqu'à la racine de l'arbre, en appliquant aux coordonnées des sommets l'opération correspondant à chacun des groupes de transformations rencontrés. Les transformations sont donc appliquées dans l'ordre, *d'une feuille vers la racine*.

L'arbre d'une scène 3D peut contenir des groupes de transformations et des formes 3D, mais aussi d'autres types d'éléments utilisés par Java 3D : des noeuds représentant des lumières, des fonds d'écran, des comportements (animation et réaction aux événements), des sources de sons,...
Les classes représentant un groupe dérivent des classes javax.media.j3d.Node et javax.media.j3d.Group.
Les classes représentant une feuille dérivent des classes javax.media.j3d.Node et javax.media.j3d.Leaf.

java.lang.Object

javax.media.j3d.SceneGraphObject

javax.media.j3d.Node

javax.media.j3d.Group

javax.media.j3d.BranchGroup

javax.media.j3d.OrderedGroup

javax.media.j3d.SharedGroup

javax.media.j3d.Switch

javax.media.j3d.TransformGroup

javax.media.j3d.Leaf

javax.media.j3d.Background

javax.media.j3d.Behavior

javax.media.j3d.BoundingLeaf

javax.media.j3d.Clip

javax.media.j3d.Fog

javax.media.j3d.Light

javax.media.j3d.Link

javax.media.j3d.Morph

javax.media.j3d.Shape3D

javax.media.j3d.Sound

javax.media.j3d.Soundscape

javax.media.j3d.ViewPlatform

javax.media.j3d.NodeComponent

...

Globalement, une scène 3D est documentée par la représentation graphique de son arbre.

La liste des classes de Java3D est très grande, toutefois le lecteur intéressé peut consulter le site officiel de java pour de plus amples détails.

Table des figures

Bibliographie et « wébographie »

(Joseph GABAY, ***Merise et UML pour la modélisation des systèmes d'information*****,** 2001) pour la modélisation avec UML

(Ed TITLE, ***Schaum's Outline Of XML,*** 2003) pour les schéma XML

(Pierre Alain MULLER, ***Modélisation objet avec UML,*** 1997) pour la modélisation

(Jean Pierre MULLER, 2004) MIMOSA : ***Représentation des connaissances et simulation***

(Jean Pierre MULLER, 2003) MIMOSA : ***Les dernières évolutions de MIMOSA***

(Olivier SIGAUD, ***Introduction à la modélisation objet avec UML***) pour la modélisation avec UML

(F. Bousquet, J-P. Mûller, C. Le Page, ***Modélisation et simulation multi agents***) pour la simulation

(http://www.cirad.fr) pour CORMAS

(**http://www.omg.org**) pour la modélisation avec le langage UML

(**http://www.madkit.org**) pour MADKIT

(**http://repast.sourceforge.net**) pour REPAST

(**http://java.sun.com**) pour la programmation

(**http://www.eteks.com**) pour JAVA3D

(**http://mimosa.sourceforge.net**) site de la communauté des développeurs de MIMOSA

(**http://opengl.org**) pour le moteur OPENGL

(**http://developpez.com**) pour XML

(**http://jdom.org**) pour XML

(**http://www.inria.fr**) pour information sur les gros projets informatiques

http://www.cnrs.fr pour information sur les gros projets informatiques

Tables des index

—T—

—U—

—V—

—W—

—X—